PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 13/00	A1	(11) International Publication Number: WO 99/28827 (43) International Publication Date: 10 June 1999 (10.06.99)

(21) International Application Number: PCT/US98/25760

(22) International Filing Date: 3 December 1998 (03.12.98)

(30) Priority Data:
60/067,224 3 December 1997 (03.12.97) US

(71) Applicant: BELL COMMUNICATIONS RESEARCH, INC. [US/US]; 445 South Street, Morristown, NJ 07960–6438 (US).

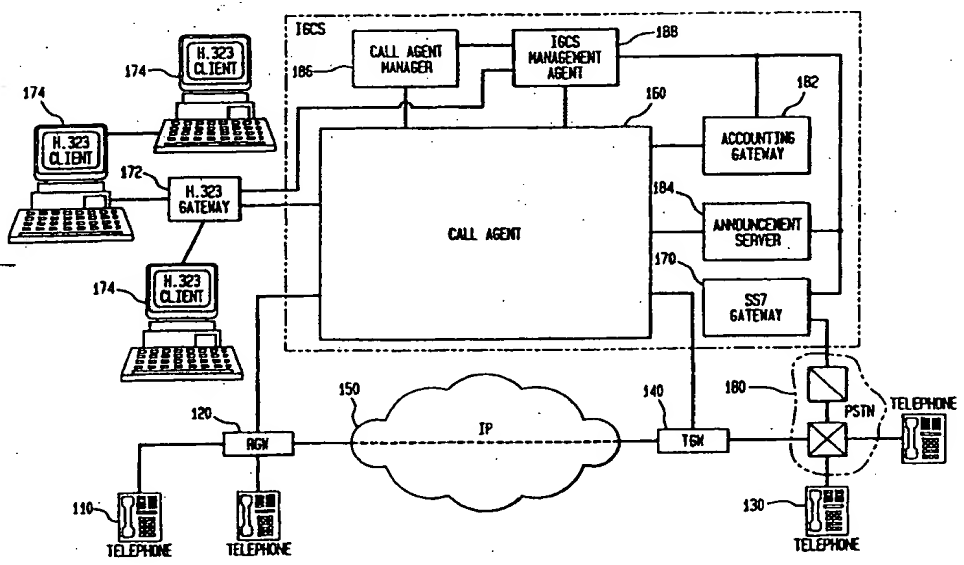
(72) Inventors: ARANGO, Mauricio; Apartment 26, 10 Ridgedale Avenue, Madison, NJ 07940 (US). CAHL, Louis; 445 South Street, Morristown, NJ 07960 (US). COOK, Michael; 445 South Street, Morristown, NJ 07960 (US). ELY, Thomas, Chambers; 1178 Delaware Drive, Bridgewater, NJ 08807 (US). HUITEMA, Christian; Apartment 119N, 77 Bleecker Street, New York, NY 10012 (US). OBROCK, Frederick; 445 South Street, Morristown, NJ 07960 (US). SMYK, Darek, A.; 15 Zirkel Avenue, Piscataway, NJ 08854 (US).

(74) Agents: YEADON, Loria, B. et al.; International Coordinator, Rm. 1G112R, 445 South Street, Morristown, NJ 07960–6438 (US).

(81) Designated States: CA, JP, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR MEDIA CONNECTIVITY OVER A PACKET–BASED NETWORK



(57) Abstract

Methods and systems for a distributed scalable hardware independent system that supports multiple functions regarding management (186, 188) and support (182, 184) of communications over a packet–based network. The communications supported by these methods and systems include, but are not limited to, Voice Over Internet Protocol ("VOIP") (150), voice over Asynchronous Transfer Mode ("ATM"), video conferencing, data transfer, telephony (130), and downloading video or other data (174). These methods and systems use a call agent (160), which is composed of various objects distributed along a CORBA software bus, for exercising call management over two endpoints communicating over a packet–based network.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD AND SYSTEM FOR MEDIA CONNECTIVITY OVER A PACKET-BASED NETWORK

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No.60/067,224, filed December 3, 1997, the contents of which are hereby incorporated by reference.

BACKGROUND OF THE INVENTION

The present invention relates generally to communications, and more particularly, to a method and system for managing media sessions.

The telecommunications industry is pushing to develop effective systems for implementing voice-based communications over packet-based networks, particularly voice over Internet Protocol ("IP"). The H.323 protocol standards represent one such attempt, but suffer from several disadvantages. In particular, these standards require a logical data connection between network elements, which limits flexibility, scalability, and efficiency.

Therefore, it is desirable to have a method and system for overcoming the disadvantages of conventional voice over packet-based network systems.

DESCRIPTION OF THE INVENTION

Accordingly, the present invention is directed to a communication system that substantially obviates one or more of the problems due to limitations and disadvantages of the prior art.

In accordance with the purposes of the invention, as embodied and broadly described herein, the invention comprises a packet-based network, a first subscriber unit, a first media control device connecting the first subscriber unit to the packet-based network, a second subscriber unit, a second media control device connecting the second subscriber unit to the packet-based network, and a call agent. The call agent of this embodiment is a device for managing communications between the first and second subscriber units over the network, and a device for sending and/or receiving SS7 signaling information.

In another aspect, the invention comprises a first subscriber unit coupled to a network through a first media control device, a second subscriber unit coupled to the network through a second media control device, and a call agent. The call agent of this embodiment includes a first call agent cluster coupled to the first subscriber unit through a media control device. The first call agent cluster includes a device for translating information received from the first media control device in a first protocol into a common protocol, a device for communicating with a second call agent cluster using the common protocol, a device for translating the information in the common protocol into the first protocol, and a device for controlling the first media control device for managing a media session between the first subscriber unit and the second subscriber unit over the network.

In another aspect, the invention comprises a method of managing communications between a first subscriber unit and a second subscriber unit over a network, wherein this method includes the call agent sending and/or receiving SS7 signaling information regarding management of communications over a packet-based network, the call agent managing communications between the first and second subscriber units over the network, and the first and second subscriber units communicating over the network.

In another aspect, the invention comprises a method of managing communications between a first subscriber unit and a second subscriber unit. This method comprises the steps of a first media control device coupled to the first subscriber unit transmitting information in a first protocol to a first call agent cluster regarding establishing a media session with the second subscriber unit over a packet-based network. The first call agent cluster translates the information in the first protocol to a common protocol and sets up a connection between the first call agent cluster and a second call agent cluster. The first call agent cluster and the second call agent cluster exchange information using the common protocol, the first call agent cluster translating information in the common protocol to the first protocol. The first call agent cluster transmits the information in the first protocol to the first media control device coupled to the first subscriber unit. The second call agent cluster translates information in the common protocol to a second protocol, and transmits the information in the second protocol to a second media control device coupled to the second subscriber unit. The first subscriber unit and the second subscriber unit then exchange information over the network.

The description of the invention and the following description for carrying out the best mode of the invention should not restrict the scope of the claimed invention. Both provide examples and explanations to enable others to practice the invention. The accompanying drawings, which form part of the description for carrying out the best mode of the invention, show several embodiments of the invention, and together with the description, explain the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of the IGCS system according to one embodiment of the invention.

Fig. 2 is a diagram of a call agent according to one embodiment of the invention.

Fig. 3 shows connectivity between two TGWs over a packet-based network according to one embodiment of the invention.

Fig. 4 shows connectivity between a RGW and a TGW over a packet-based network according to one embodiment of the invention.

Fig. 5 shows connectivity between two RGWs over a packet-based network according to one embodiment of the invention.

Fig. 6 is a diagram of a call agent cluster for RGW connection management and a call agent cluster for TGW connection management according to one embodiment of the invention.

Fig. 7 is a diagram of call models supported by call agent clusters according to one embodiment of the invention.

Fig. 8 is a diagram of call models supported by a traditional switch.

Fig. 9 is a flow diagram for RGW - RGW connection set-up according to one embodiment of the invention.

Fig. 10 is a flow diagram for the call agent cluster supporting RGW - RGW connectivity according to one embodiment of the invention.

Fig. 11 is a flow diagram for RGW - RGW connection tear down according to one embodiment of the invention.

Fig. 12 is a flow diagram for TGW - TGW connection set-up according to one embodiment of the invention.

Fig. 13 is a flow diagram for TGW - TGW connection tear down according to one embodiment of the invention.

Fig. 14 is a flow diagram for RGW - TGW connection set-up according to one embodiment of the invention.

Fig. 15 is a flow diagram for RGW - TGW connection tear down according to one embodiment of the invention.

Fig. 16 is a flow diagram for TGW - RGW connection set-up according to one embodiment of the invention.

Fig. 17 is a flow diagram for TGW - RGW tear down according to one embodiment of the invention.

Fig. 18 is a flow diagram of a service broker for connection set-up according to one embodiment of the invention.

BEST MODE FOR CARRYING OUT THE INVENTION

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. Wherever possible, the same reference numbers will be used throughout the drawings to refer to the same or like parts.

In a preferred embodiment of the invention, an Internet Gateway Call Server ("IGCS") is a distributed scalable hardware independent system that supports multiple functions regarding management and support of communications over a packet-based network. The communications supported by the IGCS include, but are not limited to, Voice Over Internet Protocol ("VOIP"), voice over Asynchronous Transfer Mode ("ATM"), video conferencing, data transfer, telephony, and downloading video or other data. These communications will be referred to as media sessions.

To accommodate various possible future requirements, the IGCS may be divided into separate components, each of which may or may not be present in a particular IGCS deployment. As shown in Fig. 1, these components include a call agent 160, SS7 gateways 170, an accounting gateway 182, and an announcement server 184. The SS7 gateway 170 within a preferred embodiment of the IGCS allows the IGCS to attach to and be part of the existing PSTN while other components within the system may interface with packet-based media devices. In a preferred embodiment of this system, a call agent 160 sets-up a connection between subscriber units 110 and 130 over a network 150. In a preferred embodiment, the subscriber units 110 and 130 are telephones, and the network is an IP network 150. Although, the invention is not limited to this application, and the subscriber units can be any user device for sending and receiving information. Also, the network can be any packet-based network capable of carrying data information, including an ATM network.

Media control devices 120 and 140 each connect a subscriber unit, 110 or 130, respectively, to the network 150. For supporting voice over Internet protocol (VOIP) communications, the media control devices 120 and 140 are termed VOIP gateways and in a preferred embodiment can be either a Trunking Gateway ("TGW") 140 or a residential gateway ("RGW") 120. A TGW 140 connects a Public Switched Telephone Network ("PSTN") 180 to the network

150, and thus provides the subscriber unit 130 with a connection to the network 150.

For this type of connectivity, signaling information, such as call-set up, tear-down, and management signaling, i.e., SS7 signaling, is sent through the PSTN 180 to an SS7 gateway 170, which connects the SS7 signaling information to the call agent 160. The call agent uses this information to set-up, tear-down, or manage the connection by sending messages to the TGW 140. In a preferred embodiment, these messages are Simple Gateway Control Protocol ("SGCP") messages, however, other protocols may be supported depending on the type of media control device the call agent is supporting.

After the call agent 160 sets-up a connection for the subscriber unit 130 over the network 150, information is exchanged between the subscriber units 110 and 130 over the network through their respective gateways 120 and 140. Thus, in a preferred embodiment, the call agent 160 is used for call management and the information exchanged between the subscriber units 110 and 130 does not pass through the call agent 160. In a preferred embodiment, the media control devices use Real Time Protocol ("RTP") and Real Time Control Protocol ("RTCP") to communicate over an IP network.

In another preferred embodiment, the media control devices use an appropriate ATM Adaptation Layer ("AAL") type to communicate over an ATM network.

An RGW 120 provides a traditional analog interface to the network. RGWs may include "set-top boxes." Unlike a TGW, RGWs both send and receive signaling information to/from the call agent 160.

In addition, the call agent of a preferred embodiment can communicate with various media control devices controlling various other types of subscriber units. For example, as shown in Fig. 1, an H.323 gateway 172 may be used as a media control device to provide an interface between the call agent and H.323 clients 174.

The call agent objects, illustrated in Fig. 2, consist of call agent clusters 210, an ingress service broker 220, an egress service broker 230, and a network resource database 240. These objects are distributed along a Common Object Request Broker Architecture ("CORBA") software bus 250. CORBA allows applications to communicate with one another no matter where they are located or who has designed them, thus allowing for flexible placement of them to suit considerations of cost, performance, and availability.

Call agent clusters are logical groupings of call agent components, and handle the specifics of call management. Their two central functions are exercising control over a media control device, which in a preferred embodiment is a VOIP gateway, and translating messages from one protocol, such as SGCP or ISDN User Part ("ISUP"), into a protocol that is common to all objects within the call agent. In a preferred embodiment, this common protocol is the Multi Call Agent Protocol ("MCAP") developed by Bellcore, which is defined using the CORBA Interface Definition Language ("IDL"). A script for this protocol is provided in Appendix A.

The detailed operation of a call agent cluster varies depending on the type of media control device it manages. The operations of call agent clusters for managing RGWs and TGWs are discussed later.

In a preferred embodiment, there are three possible connection types between subscriber units where the subscriber unit is connected to the network via a TGW or RGW. The first connection type is where both subscriber units are connected to the network via a TGW, as illustrated in Fig. 3. The second is where one subscriber unit is connected to the network via an RGW and another is connected to the network via a TGW, as illustrated in Fig. 4. The third connection type is where both subscriber units are connected to the network via RGWs as illustrated in Fig. 5. These figures are discussed in more detail below.

Fig. 3 illustrates the relevant system components for supporting communications between two subscriber units both connected to the network 150 via a TGW (a TGW-TGW connection). These components preferably include TGWs 310 and 312, an ingress call agent cluster 314, an egress call agent cluster 316, an ingress service broker 318, an egress service broker 320, a network resource database 322, SS7 gateways 326 and 328, and a CORBA software bus 324. Flow diagrams for connection set-up and tear down for this type of connection are shown in Figs. 12 and 13, respectively, which are discussed later.

Fig. 4 illustrates the relevant system components for supporting communications between two subscriber units where one is connected to the network 150 via an RGW and the other via a TGW. For this embodiment the relevant components include an RGW 410, a TGW 412, an ingress call agent cluster 414, an egress call agent cluster 416, an ingress service broker 418, an egress service broker 420, a network resource database 422, and a CORBA software bus 424. Flow diagrams for connection set-up and tear down for this type of connection are shown in Figs. 14 and 15, respectively, which are discussed later.

Fig. 5 illustrates the relevant components where both subscriber units are connected to the 150 network via an RGW. These components preferably include RGWs 510 and 512, an ingress call agent cluster 514, an egress call agent cluster 516, an ingress service broker 518, an egress service broker 520, a network resource database 522, and a CORBA software bus 524. Flow diagrams for connection set-up and tear down for this type of connection are shown in Figs. 9 and 11, respectively, which are discussed later.

The objects comprising a call agent cluster vary depending on the type of media control device managed by the cluster. Fig. 6 provides a top level diagram of the objects of a generic call agent cluster for managing a TGW 640 and a call agent cluster for managing an RGW 660. These objects include a message queue 610 and 620, an endpoint manager 614 and 624, a state machine 616 and 626, and a media control device manager 618 and 628. In addition, the call agent cluster may also contain a message handler 612, which is used in TGW connection management. These components are distributed along a CORBA software bus 630.

The message queue 610 and 620 of a call agent cluster temporarily stores messages received from a media control device 426 or 410, respectively. Each call agent cluster preferably contains at least one message queue, and different queues are used for managing different types of media control devices. For example, there are different message queues for RGW and TGW connection management. The message queue for TGW connection management is referred to as an ISUP message queue, and the message queue for RGW connection management is referred to as an SGCP message queue.

The operation of a message queue for TGW connection management consists of an SS7 gateway 426 sending ISUP messages to the queue 610. The queue stores the messages and then forwards them to a message handler 612 on a first in first out basis. For RGW connection management, an RGW 410 sends SGCP messages to the queue 620. The messages are stored and then transmitted directly to an endpoint manager 624. As such, a call agent cluster for RGW connection management need not contain a message handler.

The endpoint manager 614 and 624 is responsible for managing the state of each call, and each call agent cluster contains at least one. The endpoint manager 614 and 624 has two principal functions. The first is receiving messages from the various components of the system, such as message queues 610 and 620, service brokers 418 and 420 and state machines of peer call agent clusters 616 and 626.

The second principal function of the endpoint manager 614 and 624 is storing information on the state of each connection. The endpoint manager 614 and 624 preferably stores this information in a construct called the connection set descriptor. This construct is sufficiently generic to contain information associated with the various possible types of endpoints, such as TGWs and RGWs. The contents of the connection set descriptor for the preferred embodiment are illustrated in the following table:

Field	Data	Description
Source State	Enumeration	An enumeration of states indicating the cal source status; e.g. for SS7, a blocked circu
Connection	Call ID	A unique value used to correlate descripto between the source and target Call Agents
Source Endpoint	Enumeration	An enumeration of state indicating the call status, e.g., for SS7, waiting for Address Complete Message ("ACM").
	Timestamp	Used by an independent thread within the Endpoint Manager to determine if the allotted time has expired for next state change; e.g. for SS7, the receipt of an ACI in response to sending an Initial Address Message ("IAM").
	Media Control Device Information	Hardware specific source media control device information; e.g., the compression algorithm in use.
Target Endpoint	CIC Telephone # Announcement ID	Source ID of target endpoint.
	IOR	Target Endpoint Manager Interoperable Object Reference ("IOR").
	Telephone #	Telephone number associated with the target endpoint.
	Media Control Device Information	Hardware specific target gateway information, e.g., the compression algorith in use.

The above summary serves as a guideline that can be specialized for use with specific types of endpoints.

The call agent cluster preferably stores these connection set descriptors in a connection set descriptor manager 670 and 680, an independent object visible only to the endpoint manager. In a preferred embodiment, the storage is in memory with backups written to disk, and there is one connection set descriptor manager per endpoint manager. Although, in other embodiments, there can be one per call agent cluster.

The endpoint manager 614 and 624 upon receiving a message, determines the connection set descriptor associated with the connection, determines the appropriate state machine and then forwards both the connection set descriptor and message to the state machine 616 and 626.

State machines 616 and 626, based on the received message and connection set descriptor, determine an associated action (transition) to take using a call model. The call model used by the state machine 616 and 626 depends upon the type of media control device that the call agent cluster exercises control over. For example, a call agent cluster uses an SGCP ingress/egress call model for RGW connection management, while an ISUP call model would be used for TGW connection management. Appendix B provides a call model script for a preferred embodiment.

As shown in Fig. 7, each call agent cluster preferably supports a half call model on either the ingress or egress side of the call. That is, the ingress call agent cluster 710 supports an ingress call model 720, and the egress call agent cluster 730 supports an egress call model 740. In contrast, as shown in Fig. 8, in traditional telephony, both the ingress switch 810 and egress switch 830 support both an ingress call model 820 and 840 and an egress call model 822 and 842, respectively.

After the action is determined, it is taken. This action can involve a sequence of interactions and include transmitting messages to a gateway manager 618 and 628, service broker 418, or an endpoint manager of a peer call agent cluster 614 and 624. In a preferred embodiment, these messages are transmitted over a CORBA software bus using the MCAP protocol.

The state machine can be described as "stateless," meaning that the state machine has no independent knowledge of the state of a connection. It preferably receives this information from the endpoint manager as part of the connection set descriptor. This permits an endpoint manager to work with a number of different state machines over the course of a connection for management purposes. In addition, it permits the endpoint manager to work with the state machines of different network service providers that may perform unique functions.

The media control device manager 618 and 628 preferably interacts with the state machine 616 and 626 to manage the respective media control device. It accomplishes this by receiving MCAP messages from the state manager 616 and 626, translating them to the appropriate protocol, and transmitting SGCP messages to the respective media control device 410 or 412, respectively.

In addition to the above components, the call agent cluster may also contain a message handler. This component is preferably used for TGW connection management, and there is no counterpart for RGW connection management. The principal function of the message handler is determining which of a plurality of endpoint managers should service the call. Thus, the message handler receives an ISUP message from the message queue, determines which endpoint manager should receive the message, and forwards the message to this endpoint manager.

From an Object Oriented ("OO") perspective, the implementation of similar objects, e.g. the RGW and TGW Message Queues, are candidates for inheritance, meaning the objects inherent to the call agent cluster are designed to be generic in structure and can be reused for handling different protocols. In this way, the implementation can take advantage of the benefits gained from identifying common behavior and design patterns.

Fig. 9 provides a flow diagram for describing the operation of the call agent for setting up a connection between end-users connected to the network via RGWs, such as illustrated in Fig. 5. The process is initialized by the RGW (Step 101). The ingress call agent cluster and the RGW then exchange several messages (Step 102). These messages may include messages to play a dial-tone, collect digits, and enter receive mode. The ingress call agent cluster then sends an MCAP message to the egress call agent cluster regarding setting up a connection between the call agent clusters (Step 103). The internal operations of the call agent clusters are discussed later.

The egress call agent cluster then instructs, using SGCP, the RGW to setup a connection in send/receive mode and start a ringing signal in the subscriber unit (Step 104). After which, the egress call agent cluster sends an MCAP message to the ingress call agent cluster indicating that it created a connection (Step 105). The ingress call agent cluster then instructs, using SGCP, the RGW to start a ringing tone in the subscriber unit (Step 106). When the call is answered, the RGW sends an off-hook message to the egress call agent cluster (Step 107), which is forwarded, using MCAP, to the ingress call agent cluster (Step 108). After which, the ingress call agent cluster, using SGCP, instructs the RGW to enter send/receive mode (Step 109).

Fig. 10, provides a more detailed flow diagram for describing the internal operations of the ingress call agent cluster for the above described RGW to RGW connection set-up. An RGW sends an SGCP message to the message queue of the ingress call agent cluster indicating that it wishes to establish a connection with a second media control device (Step 201). This message is placed in the queue.

The message is then passed to the endpoint manager (Step 202) on a first in first out of the queue basis. The endpoint manager then transmits the connection set descriptor and message to the state machine (Step 203).

The state machine then uses the received message and connection set descriptor to take a specified action, determined by the applicable call model. In this case, the specified action is sending an MCAP message to the endpoint manager of the egress call agent cluster (Step 204). It should be noted that a service broker is used to establish the connection between call agent clusters; the operations of this process are discussed later.

The state machine of the egress call agent cluster then sends an MCAP message to the endpoint manager of the ingress call agent cluster indicating that it set-up a connection with the RGW (Step 205). The endpoint manager forwards this message and the connection set descriptor to the state machine (Step 206). The state machine determines which action to take using this received information and the applicable call model. In this case, the state machine sends an MCAP message to the gateway manager instructing it to instruct the RGW gateway to start ringing (Step 207). After which, the gateway manager sends an SGCP message to the RGW to start ringing (Step 208).

When the call is answered, the state machine of the egress call agent cluster sends a message to the endpoint manager of the ingress call agent cluster indicating that the phone on the egress side is off-hook (Step 209). The endpoint manager then forwards this message along with the associated connection set descriptor to the state machine (Step 210).

The state machine then determines the action to take using this received information. In this case, the state machine transmits an MCAP message to the gateway manager indicating that the phone has been answered (Step 211). The gateway manager then forwards, using SGCP, this message to the RGW (Step 212).

In a preferred embodiment, call agent cluster objects may be shadowed. For example, a call agent cluster can contain an idle second state machine for use in the event the first state machine fails. Because a CORBA bus is used in the preferred embodiment and state machines are stateless, this second state machine need not share the same hardware environment as the primary object.

Fig. 11 provides a flow diagram for tearing down a connection between RGWs. The process is initialized when an RGW sends an on-hook message to the ingress call agent cluster (Step 301), which is received by the message queue of the call agent cluster and forwarded to the state machine via the endpoint manager. The ingress call agent cluster then instructs the RGW to tear down the connection (Step 302). After which, the ingress call agent cluster sends an MCAP message to the egress call agent cluster (Step 303), which instructs, using SGCP, its RGW to tear down the connection (Step 304). The egress call agent cluster then sends a message to the ingress call agent cluster indicating that the above action was taken (Step 305).

Fig. 12 provides a flow diagram for setting up a TGW to TGW connection. The process is initialized by a switch sending an Initial Address Message ("IAM") message to the ingress call agent cluster indicating it wishes to establish a media session between two subscriber units (Step 401). The ingress call agent cluster then instructs, using SGCP, the TGW on the ingress side to set up a connection in receive mode. (Step 402) The ingress call agent then forwards an MCAP message indicating this information to the egress call agent cluster (Step 403). After which, the egress call agent cluster instructs, using SGCP, the TGW to set-up a connection in send/receive mode (Step 404). The egress call agent cluster then sends an IAM message to the switch (Step 405). After which, the switch sends an Address Complete Message ("ACM") to the egress call agent cluster (Step 406). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that it took the requested action (Step 407). The ingress call agent cluster then sends an ACM to the switch (Step 408). After which, the switch sends an Answer Message ("ANM") to the egress call agent cluster (Step 409). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that the call has been answered (Step 410). After which, the ingress call agent cluster instructs the TGW to enter send/receive mode (Step 411). The ingress call agent cluster then sends an ANM message to the switch (Step 412).

Fig. 13 provides a flow diagram for tearing down a TGW to TGW connection. The process is initialized by a switch sending a Release Message ("REL") to the ingress call agent cluster (Step 501). The ingress call agent cluster then instructs the TGW to tear down the connection (Step 502). After which, the ingress call agent cluster sends an MCAP message to the egress call agent cluster (Step 503). The egress call agent cluster then instructs the TGW to tear down the connection (Step 504). The egress call agent cluster then sends an REL to the switch (Step 505). The switch then sends a Release Confirm ("RLC") to the egress call agent cluster (Step 506). The egress call agent cluster sends an MCAP message to the ingress call agent cluster indicating that the connection has been released (Step 507). After which, the ingress call agent cluster sends an RLC message to the switch.

Fig. 14, provides a flow diagram for setting-up a connection between an RGW and a TGW. The process is initialized by the RGW and ingress call agent cluster exchanging several SGCP messages relating to setting up a connection. (Step 601). These messages include messages related to playing a dial-tone, collecting digits and entering receive mode. The ingress call agent cluster then sends an MCAP message to the egress call agent cluster indicating it wishes to set-up a media session (Step 602). The egress call agent cluster then instructs the TGW to set up a connection in send/receive mode (Step 603). After which, the egress call agent cluster constructs an IAM and sends it to the switch (Step 604). The switch then sends an ACM to the egress call agent cluster (Step 605). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that it took the above action (Step 606). The ingress call agent cluster then instructs the RGW to start a ringing tone (Step 607). The switch then sends an ANM to the egress call agent cluster (Step 608). After which, the egress call agent cluster sends an MCAP message indicating that the call was answered to the ingress call agent cluster (Step 609). The ingress call agent cluster then instructs the RGW to enter send/receive mode (Step 610).

Fig. 15 provides a flow diagram for tearing down an RGW to TGW connection. The RGW initializes the process by sending an on-hook message to the ingress call agent cluster (Step 701), which instructs the RGW to tear down the connection (Step 702). The ingress call agent cluster then sends an MCAP message to the egress call agent cluster instructing it to tear down the connection (Step 703). After which, the egress call agent cluster instructs the TGW to tear down the connection by sending it an SGCP message (Step 704). The egress call agent cluster then constructs a REL and sends it to the switch (Step 705). After which, the switch sends an RLC to the egress call agent cluster (Step 706). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that the connection has been released (Step 707).

Fig. 16 provides a flow diagram for setting up a TGW to RGW connection. The switch initializes the process by sending an IAM to the ingress call agent cluster (Step 801). After which, the ingress call agent cluster instructs the TGW to set-up a connection in receive mode (Step 802). The ingress call agent cluster then sends an MCAP message to the egress call agent cluster regarding establishing a connection (Step 803). The egress call agent cluster then instructs the RGW to setup a connection in send/receive mode and start a ringing signal (Step 804). After which, the egress call agent cluster sends an MCAP message to the ingress call agent cluster indicating that it took the above action (Step 805). The egress call agent cluster then constructs an ACM and sends it to the switch (Step 806). The RGW then sends an off-hook message to the egress call agent cluster (Step 807). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that the call was answered. (Step 808). The ingress call agent cluster then instructs the TGW to enter send/receive mode (Step 809). After which, the ingress call agent cluster constructs an ANM and sends it to the switch (Step 810).

Fig. 17, provides a flow diagram for tearing down a TGW to RGW connection. The switch initializes the process by sending a REL to the ingress call agent cluster (Step 901). The ingress call agent cluster then instructs the TGW to tear down the connection (Step 902). The ingress call agent cluster then sends an MCAP message to the egress call agent cluster indicating that the connection is to be torn down. (Step 903). The egress call agent cluster then instructs the TGW to tear down the connection (Step 904). The egress call agent cluster then sends an MCAP message to the ingress call agent cluster indicating that the connection was released (Step 905). After which, the ingress call agent cluster constructs an RLC and sends it to the switch (Step 906).

The call agent uses a service broker for establishing communications between an ingress and egress call agent cluster. When a subscriber unit wishes to establish communications with another subscriber unit, the ingress call agent cluster forwards the information to an ingress service broker. The ingress service broker performs an algorithm for determining the egress service broker for the egress call agent cluster of the subscriber unit it wishes to establish communications with. The egress service broker then performs an algorithm for determining the appropriate egress call agent cluster. The service brokers use routing engines for performing these algorithms, which use a routing table, a look-up table, for determining the appropriate information. In addition, the routing performs the functions of digit translation and classifying the connection (e.g., is it a 800 call or long distance call).

In a preferred embodiment, the proper execution of the routing engine is assured by, during initialization, loading into a table in a high speed database the egress call agent cluster's endpoint manager's Interoperable Object Reference ("IOR"). In a preferred embodiment, the table points to a relationship between a given endpoint manager's IOR and a route path for a media session. The routing engine consults with this table and, depending on the number dialed, selects the appropriated endpoint manager's IOR and routes MCAP messages to it to establish a connection.

Fig. 18 provides a flow diagram for this process. Once a call agent cluster receives a request from a subscriber unit to establish communications with a second subscriber unit, the message is forwarded from an endpoint manager to the state machine, as discussed above (Step 1001). The state machine then forwards this request using MCAP to the ingress service broker (Step 1002). The ingress service broker, using a routing engine, determines the appropriate egress service broker and forwards the request to it using MCAP (Step 1003). The egress service broker then, using a routing engine, determines the appropriate egress call agent cluster and, using MCAP, forwards the message to its endpoint manager (Step 1004). After which, the ingress and egress call agent clusters communicate directly with one another (Step 1005).

In a preferred embodiment, a network resource data base stores information regarding the network. This network resource database is coupled to the CORBA bus. As such, the service brokers and call agent clusters can access this database.

In addition to the above described call agent and SS7 gateways, the IGCS may include a call agent manager 186, accounting gateway 182 announcement server 184, and an IGCS management agent 188.

The accounting gateway 182 in a preferred embodiment is a media control device by which start/stop call records are disseminated to a central point where they are translated from CORBA into Remote Authentication Dial-In User Service ("RADIUS"), then translated into industry standard BAF records for distribution to back-end billing systems. As shown in Fig. 2, the accounting gateway 182, in a preferred embodiment, communicates with a call agent cluster 210 of a call agent 160.

In a preferred embodiment, the announcement server 184 is a media control device which responds to SGCP messages by playing recorded announcements. As shown in Fig. 2, the announcement server communicates with a call agent cluster 210 of a call agent 160.

The call agent manager 186 is directly responsible for the local management of the call agent, for example: displaying the state of the call agent components, displaying alarms reported by the call agent components and object control, such as shutdown.

The IGCS management agent 188 of a preferred embodiment manages all IGCS components including the call agent clusters, network resource database, accounting gateway and announcement server.

In a preferred embodiment, the overall network management software is capable of displaying usage statistics accumulated in the call agent clusters. A call agent of a preferred embodiment is designed to be Simple Network Management Protocol ("SNMP") manageable by any third party SNMP management software, e.g., HP open view. The architecture allows for SNMP sets, gets, and traps to be sent to a call agent SNMP agent whose function is the gathering of statistics via CORBA method invocations executed against call agent cluster objects. Also, SNMP sets are handled via method invocations that update data in the call agent cluster.

While it has been illustrated and described what are at present considered to be preferred embodiments and methods of the present invention, it will be understood by those skilled in the art that various changes and modifications may be made, and equivalents may be substituted for elements thereof without departing from the true scope of the invention.

In addition, many modifications may be made to adapt a particular element, technique or implementation to the teachings of the present invention without departing from the central scope of the invention. Therefore, it is intended that this invention not be limited to the particular embodiments and methods disclosed herein, but that the invention include all embodiments falling within the scope of the appended claims.

Appendix A

The Multi Call Agent Protocol (MCAP) is designed to provide two capabilities:

1. Core information, logically grouped in the following categories:

Session Management	A session ties users (calling party and called party) together, who are identified by such means as the calling number and the called number.
Connection Management	A call is composed of a set of connections, each of which ties together a pair of Call Agent Clusters. The associated resources are keyed by a globally unique ID.
Gateway Management	A Call Agent controls a specific connection on an associated gateway by exchanging information keyed by a locally unique ID.
Routing Management	Routing specific information, e.g. toll-free number translation.
Service Management	Both the calling party and the called party can have specific services with potential interactions. The Service Execution Engine identifies the associated services and determines their application.
Parameter	MCAP message specific information; e.g. version.

2. A tunnel for messages of a specific protocol that can be carried as a payload either in a native or a parsed format. The native format is the given octet stream; the parsed format is the representation of the given octet stream in the MCAP definition language.

   The following primitive message types can support these capabilities.

MCAP_CREATE	Create a new connection between Call Agent Clusters.
MCAP_EVENT	Indicate an event that occurs during the lifetime of a connection. This can serve as, for example, an acknowledgement or an indication of connection state change.
MCAP_DELETE	Delete an existing connection between Call Agent Clusters.
MCAP_TUNNEL	Tunnel, i.e. passthrough, a specific protocol message from one Call Agent Cluster to another without providing other information.

Arguably, MCAP_CREATE and MCAP_DELETE can be subsumed by MCAP_EVENT, but it is useful to distinguish these actions. MCAP_TUNNEL is provided to generically accommodate the needs of a specific protocol.

Informally, MCAP is summarized as follows:

Information Category	**MCAP Message**			
	MCAP_CREATE	**MCAP_EVENT**	**MCAP_DELETE**	**MCAP_TUNNEL**
Session	Calling Number *Calling Name* Called Number			
Connection	Call ID Return Address	Call ID Return Address	Call ID	Call ID
Gateway	Connection Descriptor	*Connection Descriptor*		
Routing	Parameter			
Service	Parameters			
Parameter	Parameter	Parameter	Parameter	
Tunnel	*Protocol Type* *Message Type* *Message*	*Protocol Type* *Message Type* Message	*Protocol Type* *Message Type* *Message*	Protocol Type Message Type Message

Notes

- Italicized items are optional; others are mandatory.
- The Call ID is a globally unique value defined by the Call Agent that is associated with the connection throughout its duration. It is used as the key to identify the associated resources.
- The return address is the handle to be used by the receiver to locate the sender in the event that subsequent messaging is needed.

- The Connection Descriptor contains the Connection ID, which is locally defined and used by the gateway, and specific Real Time Protocol (RTP) parameters; e.g. encoding method.

- The tunnel message can be in either parsed or native format.

The initial MCAP version was primarily the result of empirical work done in the development of the Call Agent. This version reflects the next step in addressing challenge presented by rapidly evolving Internet Protocol (IP) Telephony technology.

The formal MCAP reference definition uses the Common Object Request Broker Architecture (CORBA) 2.0 [1] [2] Interface Definition Language (IDL) for the high level description and the associated Internet Inter-ORB Protocol (IIOP) for the low-level "wire" encoding. This is similar to the use of Abstract Syntax Notation One (ASN.1) [3] [4] and the companion Basic Encoding Rules (BER) [4] [5]: the protocol designer is able to described structured information in a high-level, machine independent language and then mechanically derive the actual encoding. CORBA is chosen because it is used in the implementation of the Call Agent and has widespread interest and support throughout the industry.

This version of MCAP supports the following tunnel protocols:

- Simple Gateway Control Protocol (SGCP) [6].

- Integrated Services Digital Network User Part (ISUP) [7].

The complete MCAP IDL is presented in:

- Appendix A1: *MCAP IDL*
- Appendix A2 : *VOIP Gateway IDL*
- Appendix A3: Routing IDL
- Appendix A4: Service IDL
- Appendix A5: Parameter IDL
- Appendix A6: SGCP Tunnel IDL
- Appendix A7: ISUP Tunnel IDL
- Appendix A8: ISUP Message IDL

References

[1] *CORBA: Architecture and Specification*, OMG, 1997.

This covers CORBA V 2.0, which is commonly supported in current implementations; e.g. INPRISE VisiBroker; V3.2 (Java). OMG has published V2.2.

[2] A. Vogel & K. Duddy, *Java Programming with CORBA*, 2nd edition, John Wiley & Sons, 1998.

[3] Recommendation X.208, *Open systems interconnection: specification of Abstract Syntax Notation (ASN.1)*, CCITT Blue Book, Fascicle VII.4, ITU, 1989, pp. 57-130.

[4] D. Steedman, *Abstract Syntax Notation One (ASN.1), the Tutorial and Reference*, Technology Appraisals, 1993.

[5] Recommendation X.209, *Open systems interconnection: specification of Basic Encoding Rules for Abstract Syntax Notation (ASN.1)*, CCITT Blue Book, Fascicle VII.4, ITU, 1989, pp. 131-151.

[6] Arango, M., Huitema, C., *Simple Gateway Control Protocol (SGCP), Version 1.0*, May 15 '98.

[7] GR-317-CORE, *Switching Systems Generic Requirements for Call Control Using the Integrated Services Digital Network User Part (ISDN)*, Issue 2, Dec. '97.

Appendix A1: MCAP IDL

```
#include "mcapVoipGateway.idl"
#include "mcapRouting.idl"
#include "mcapService.idl"
#include "mcapParameter.idl"
#include "mcapSgcpTunnel.idl"
#include "mcapIsupTunnel.idl"

module Mcap {
   //
   // MCAP version
   //

   const string version = "2.0";

   //
   // tunnel message definition
   //

   enum ProtocolType {
      SGCP,
      ISUP
   };

   union TunnelMessage switch (ProtocolType) {
      case SGCP:  McapSgcpTunnel::SgcpTunnelMessage
sgcpTunnelMessage;
      case ISUP:  McapIsupTunnel::IsupTunnelMessage
isupTunnelMessage;
   };

   union OptionalTunnelMessage switch (boolean) {
      case TRUE:  TunnelMessage   tunnelMessage;
   };

   //
   // MCAP message definition
   //

   union OptionalCallingName switch (boolean) {
      case TRUE:  string  callingName;
   };

   union OptionalCallingEmailAddress switch (boolean) {
      case TRUE:  string  callingEmailAddress;
   };

   union OptionalConnectionDescriptor switch (boolean) {
        case TRUE: McapVoipGateway::ConnectionDescriptor
connectionDescriptor;
    };

    typedef sequence <McapService::ServiceParameter>
ServiceParameters;

    struct McapCreateMessage {
        // version
        string                      version;

        // session data
        string                      callingNumber;
        OptionalCallingName              callingName;
        string                      calledNumber;

        // connection data
        string                      callId;
        Object                       returnAddress;

        // gateway data
        McapVoipGateway::ConnectionDescriptor   connectionDescriptor;

        // no routing data

        // service data
        ServiceParameters                   serviceParameters;

        // parameter data
        McapParameter::CreateParameter        createParameter;

        // tunnel data
        OptionalTunnelMessage                 tunnelMessage;
    };

—   struct McapEventMessage {
        // version
        string                      version;

        // no session data

        // connection data
        string                      callId;
        Object                       returnAddress;

        // gateway data
        OptionalConnectionDescriptor          connectionDescriptor;

        // no routing data
    // no service data

    // parameter data
    McapParameter::EventParameter        eventParameter;

    // tunnel data
    OptionalTunnelMessage              tunnelMessage;
};

struct McapDeleteMessage {
    // version
    string                        version;

    // no session data

    // connection data
    string                        callId;

    // no gateway data

    // no routing data

    // no service data

    // parameter data
    McapParameter::DeleteParameter       deleteParameter;

    // tunnel data
    OptionalTunnelMessage              tunnelMessage;
};

struct McapTunnelMessage {
    // version
    string                        version;

    // no session data

    // connection data
    string                        callId;

    // no gateway data

    // no routing data

    // no service data

    // no parameter data

    // tunnel data
    TunnelMessage                    tunnelMessage;
```

```
        };

        interface McapListener {
           void mcapCreate(in McapCreateMessage msg);
           void mcapEvent(in McapEventMessage msg);
           void mcapDelete(in McapDeleteMessage msg);
           void mcapTunnel(in McapTunnelMessage msg);
        };
    };
```

Appendix A2: VOIP Gateway IDL

```
module McapVoipGateway {
   enum AudioState {
      ON,
      OFF,
      UNDEFINED
   };

   enum Mode {
      SEND_ONLY,
      RECV_ONLY,
      SEND_RECV
   };

   union OptionalMode switch (boolean) {
      case TRUE:  Mode   mode;
   };

   struct ConnectionOptions {
      short            samplePeriod;      // msecs
      string           encodingMethod;
      AudioState           audioState;
      OptionalMode         mode;
      short            bandwidth;        // KB
   };

   struct ConnectionDescriptor {
      string           connectionId;
      ConnectionOptions  connectionOptions;
      string           sdpSessionDescriptor;
   };
};
```

Appendix A3: Routing IDL

```
module McapRouting {
   enum RoutingType {
      REQUEST,
      ANNOUNCEMENT,
      ROUTE,
      CONNECT
   };

   struct Request Data {
      string  clusterId;
      boolean onNet;
   };

   union RoutingData switch (RoutingType) {
      case REQUEST: RequestData requestData;
   };

   struct RoutingParameter {
      RoutingType type;
      RoutingData data;
   };
};
```

Appendix A4: Service IDL

```
module McapService {
   enum ServiceType {
      CALLER_ID_BLOCKING,
      CALL_FORWARDING
   };

   struct CallerIdBlockingtData {
      boolean block;
   };

   struct CallForwardingData {
      unsigned long   hopCounter;
      Object          returnAddress;
   };

   union ServiceData switch (ServiceType) {
      case CALLER_ID_BLOCKING: CallerIdBlockingtData
callerIdBlockingData;
      case CALL_FORWARDING:   CallForwardingData
callForwardingData;
   };

   struct ServiceParameter {
      ServiceType type;
      ServiceData data;
   };
};
```

Appendix A5: Parameter IDL

```
module McapParameter {
  //
  // MCAP_CREATE parameter
  //

  enum CreateType {
     CALL,
     ANNOUNCEMENT
  };

  struct AnnouncementData {
    long id;
  };

  union CreateData switch (CreateType) {
     case ANNOUNCEMENT: AnnouncementData announcementData;
  };

  struct CreateParameter {
     CreateType type;
     CreateData data;
  };

  //
  // MCAP_EVENT parameter
  //

  enum EventType {
     CREATED,
     ANSWERED,
     SUSPEND,
     RESUME,
     RELEASED
  };

  struct EventParameter {
     EventType type;
  };

  //
  // MCAP_DELETE parameter
  //

  struct DeleteParameter {
     unsigned long cause;
    };

    //
    // MCAP_TUNNEL parameter
    //

    // no parameter defined
};
```

Appendix A6: SGCP Tunnel IDL

```
module McapSgcpTunnel {
   enum SgcpMessageType {
      NULL
   };

   struct SgcpTunnelMessage {
      SgcpMessageType messageType;
   };
};
```

Appendix A7: ISUP Tunnel IDL

```
#include "mcapIsupMessage.idl"

module McapIsupTunnel {
   enum IsupMessageType {
      ACM,   ANM,   BLO,   BLA,   CCR,
      CFN,   CGB,   CGBA,  CGU,   CGUA,
      COT,   CPG,   CQM,   CQR,   CRA,
      CRM,   CVR,   CVT,   EXM,   FAC,
      FOT,   GRA,   GRS,   IAM,   INF,
      INR,   LBA,   PAM,   REL,   RES,
      RLC,   RSC,   SUS,   UBA,   UBL,
      USIS
   };

   union ParsedIsupMessage switch (IsupMessageType) {
      case ACM:   McapIsupMessage::ACMMessage    acmMessage;
      case ANM:   McapIsupMessage::ANMMessage    anmMessage;
      //  BLO    no parameters
      //  BLA    no parameters
      //  CCR    no parameters
      case CFN:   McapIsupMessage::CFNMessage    cfnMessage;
      case CGB:   McapIsupMessage::CGBMessage    cgbMessage;
      case CGBA:  McapIsupMessage::CGBAMessage   cgbaMessage;
      case CGU:   McapIsupMessage::CGUMessage    cguMessage;
      case CGUA:  McapIsupMessage::CGUAMessage   cguaMessage;
      case COT:   McapIsupMessage::COTMessage    cotMessage;
      case CPG:   McapIsupMessage::CPGMessage    cpgMessage;
      case CQM:   McapIsupMessage::CQMMessage    cqmMessage;
      case CQR:   McapIsupMessage::CQRMessage    cqrMessage;
      //  CRA    no parameters
      case CRM:   McapIsupMessage::CRMMessage    crmMessage;
      case CVR:   McapIsupMessage::CVRMessage    cvrMessage;
      //  CVR    no parameters
      case EXM:   McapIsupMessage::EXMMessage    exmMessage;
      case FOT:   McapIsupMessage::FOTMessage    fotMessage;
      //  FAC    no parameters
      case GRA:   McapIsupMessage::GRAMessage    graMessage;
      case GRS:   McapIsupMessage::GRSMessage    grsMessage;
      case IAM:   McapIsupMessage::IAMMessage    iamMessage;
      case INF:   McapIsupMessage::INFMessage    infMessage;
      case INR:   McapIsupMessage::INRMessage    inrMessage;
      //  LBA    no parameters
      //  PAM    no parameters
      case REL:   McapIsupMessage::RELMessage    relMessage;
      case RES:   McapIsupMessage::RESMessage    resMessage;
```

```
        //  RLC    no parameters
        //  RSC    no parameters
        case SUS:  McapIsupMessage::SUSMessage    susMessage;
        //  UBA    no parameters
        //  UBL    no parameters
        //  USIS   no parameters
     };

     typedef sequence<octet> UnparsedIsupMessage;

     union IsupMessage switch (boolean) {
        case TRUE:  ParsedIsupMessage      parsedIsupMessage;
        case FALSE: UnparsedIsupMessage    unparsedIsupMessage;
     };

     struct IsupTunnelMessage {
        IsupMessageType messageType;
        IsupMessage     message;
     };
  };
```

Appendix A8: ISUP Message IDL

```
module IsupMessage {

 //
 //
 // Beginning of ISUP Parameter Definition Section
 //
 //

 typedef sequence<octet> bytes;

 //
 // Access Transport
 //

 struct ACCESS_TRANSPORT {
  sequence<string> parmNames;
  sequence<bytes> parmValues;
  bytes accessTransport;
 };

 //
 // Automatic Congestion Level
 //

 enum AUTOMATIC_CONGESTION_LEVEL {
  SPARE,
  LEVEL1,
  LEVEL2,
  LEVEL3
 };

 //
 // Backward Call Indicator
 //

 enum CHARGE_INDICATOR {
  NO_INDICATION,
  NO_CHARGE,
  CHARGE,
  SPARE
 };

 enum CALLED_PARTY_STATUS_INDICATOR {
  NO_INDICATION,
  SUBSCRIBER_FREE,
 CONNECT_WHEN_FREE,
 EXCESSIVE_DELAY
};

enum CALLED_PARTY_CATEGORY_INDICATOR {
 NO_INDICATION,
 ORDINARY_SUBSCRIBER,
 PAY_PHONE,
 SPARE
};

enum END_TO_END_METHOD_INDICATOR {
 NO_END_TO_END_METHOD,
 PASS_ALONG_METHOD,
 SCCP_METHOD,
 PASS_ALONG_AND_SCCP_METHOD
};

enum INTERWORKING_INDICATOR {
 NO_INTERWORKING,
 INTERWORKING
};

enum IAM_SEGMENTATION_INDICATOR {
 NO_INDICATION,
 ADDITIONAL_INFO_ADDED
};

enum ISDN_USER_PART_INDICATOR {
 ISUP_UNUSED,
 ISUP_USED
};

enum HOLDING_INDICATOR {
 HOLDING_NOT_REQUIRED,
 HOLDING_REQUIRED
};

enum ISDN_ACCESS_INDICATOR {
 TERMINATING_ACCESS_NON_ISDN,
 TERMINATING_ACCESS_ISDN,
 ORIGINATING_ACCESS_NON_ISDN,
 ORIGINATING_ACCESS_ISDN
};

enum ECHO_CONTROL_DEVICE_INDICATOR {
 INCOMING_HALF_ECHO_DEV_NOT_INCLUDED,
 INCOMING_HALF_ECHO_DEV_INCLUDED,
 OUTGOING_HALF_ECHO_DEV_NOT_INCLUDED,
 OUTGOING_HALF_ECHO_DEV_INCLUDED
```

```
};

enum SCCP_METHOD_INDICATOR {
 NO_INDICATION,
 CONNECTIONLESS,
 CONNECTION_ORIENTED,
 CONNECTIONLESS_AND_CONNECTION_ORIENTED_METHOD
};

struct BACKWARD_CALL_INDICATOR {
 CHARGE_INDICATOR chargeIndicator;
 CALLED_PARTY_STATUS_INDICATOR calledPartyStsInd;
 CALLED_PARTY_CATEGORY_INDICATOR calledPartyCatInd;
 END_TO_END_METHOD_INDICATOR endToEndMethodInd;
 INTERWORKING_INDICATOR interworkingInd;
 IAM_SEGMENTATION_INDICATOR iamSegInd;
 ISDN_USER_PART_INDICATOR isdnUserInd;
 HOLDING_INDICATOR holdingInd;
 ISDN_ACCESS_INDICATOR isdnAccessInd;
 ECHO_CONTROL_DEVICE_INDICATOR echoControlDevInd;
 SCCP_METHOD_INDICATOR sccpMethodInd;
};

//
// Business Group
//

enum PARTY_SELECTOR {
 NO_INDICATION,
 CALLING_PARTY_NUMBER,
 CALLED_PARTY_NUMBER,
 CONNECTED_PARTY_NUMBER,
 REDIRECTING_NUMBER,
 ORIGINAL_CALLED_NUMBER,
 SPARE
};

enum LINE_PRIVILEGE_INFO_IND {
 FIXED_LINE_PRIVILEGE,
 CUSTOMER_DEFINED_LINE_PRIVILEGE
};

enum BUSINESS_GROUP_ID_TYPE {
 MULTILOCATION_ID,
 INTERWORKING_ID
};

enum ATTENDANT_STATUS {
 NO_INDICATION,
 ATTENDANT_LINE
```

```
};

enum BUSINESS_GROUP_ID {
  NO_INDICATION,
  PUBLIC_NETWORK,
  NETWORK_DEPENDENT
};

enum SUBGROUP_ID {
  NO_SUBGROUP,
  SUBGROUP
};

enum TERMINATING_LINE_PRIVILEGES {
  NOTPRESENT,
  UNRESTRICTED,
  SEMIRESTRICTED,
  FULLY_RESTRICTED,
  FULLY_RESTRICTED_INTRASWITCH,
  DENIED,
  SPARE
};

enum ORIGINATING_RESTRICTIONS {
  NOTPRESENT,
  UNRESTRICTED,
  SEMIRESTRICTED,
  FULLY_RESTRICTED,
  FULLY_RESTRICTED_INTRASWITCH,
  DENIED,
  SPARE
};

struct BUSINESS_GROUP {
  PARTY_SELECTOR partySelector;
  LINE_PRIVILEGE_INFO_IND linePriInfoInd;
  BUSINESS_GROUP_ID_TYPE businessGrpIDType;
  ATTENDANT_STATUS attendantSts;
  BUSINESS_GROUP_ID businessGrpID;
  bytes BUSINESS_GROUP_ID_network_dependant;
  SUBGROUP_ID subgroupID;
  bytes SUBGROUP_ID_subgroup;
  TERMINATING_LINE_PRIVILEGES terminatingLinePri;
  ORIGINATING_RESTRICTIONS origRestriction;
  octet customer_defined_line_pri_code;
};

//
// Call Reference
//
```

```
struct CALL_REFERENCE {
 bytes CALL_IDENTITY_NUMBER;
 bytes POINT_CODE;
};

//
// Called Party Number
// Calling Party Number
//

enum NATURE_OF_ADDRESS_INDICATOR_TYPE1 {
 SUBSCRIBER_NUMBER,
 NATIONAL,
 INTERNATIONAL_NUMBER,
 ABBREVIATED_NUMBER
};

enum NATURE_OF_ADDRESS_INDICATOR_TYPE2 {
 SPARE,
 UNIQUE_SUBSCRIBER_NUMBER,
 RESERVED_FOR_NATIONAL_USE,
 UNIQUE_NATIONAL_SIG_NUMBER,
 UNIQUE_INTERNATIONAL_NUMBER,
 NONUNIQUE_SUBSCRIBER_NUMBER,
 NONUNIQUE_NATIONAL_NUMBER,
 NONUNIQUE_INTERNATIONAL_NUMBER,
 TEST_LINE_CODE,
 RESERVED_FOR_NETWORK_SPECIFIC
};

enum NATURE_OF_ADDRESS_INDICATOR_TYPE3 {
 SPRARE,
 SUBSCRIBER_NUMBER,
 RESERVED_FOR_NATIONAL_USE,
 NATIONAL_SIGNIFICANT_NUMBER,
 INTERNATIONAL_NUMBER,
 OP_REQ_SUBSCRIBER_NUMBER,
 OP_REQ_NATIONAL_NUMBER,
 OP_REQ_INTERNATIONAL_NUMBER,
 OP_REQ_NO_NUMBER_PRESENT,
 NO_NUMBER_PRESENT_CUT_THROUGH_CALL_TO_CARRIER,
 CALL_FROM_LOCAL_EXCHANGE,
 TEST_LINE_CODE,
 RESERVED_FOR_NETWORK_SPECIFIC
};

enum NATURE_OF_ADDRESS_INDICATOR_TYPE4 {
 NATIONAL_SIGNIFICANT_NUMBER
};
```

```
enum ODD_EVEN_BIT {
  EVEN,
  ODD
};

enum NUMBERING_PLAN {
  UNKNOWN,
  ISDN,
  SPARE,
  ITU_TS_DATA,
  ITU_TS_TELEX,
  PRIVATE
};

enum SCREENING {
  USER_PROVIDED_NOT_SCREENED,
  USER_PROVIDED_SCREENING_PASSED,
  USER_PROVIDED_SCREENING_FAILED,
  NETWORK_PROVIDED
};

enum PRESENTATION
{
  PRESENTATION_ALLOWED,
  PRESENTATION_RESTRICTED,
  SPARE
};

struct CALLED_PARTY_NUMBER {
  ODD_EVEN_BIT oddEvenBit;
  NATURE_OF_ADDRESS_INDICATOR_TYPE3 addressNatureInd;
  NUMBERING_PLAN numberingPlan;
  string addressSignal;
};

struct CALLING_PARTY_NUMBER {
  NATURE_OF_ADDRESS_INDICATOR_TYPE2 addressNatureInd;
  ODD_EVEN_BIT oddEvenBit;
  SCREENING screen;
  PRESENTATION presentation;
  NUMBERING_PLAN numberingPlan;
  string addressSignal;
};

//
// Calling Party Category
//

enum CALLING_PARTY_CATEGORY {
```

```
CALLING_PARTYS_CATEGORY_UNKNOWN ,
FRENCH_LANGUAGE_OPERATOR ,
ENGLISH_LANGUAGE_OPERATOR ,
GERMAN_LANGUAGE_OPERATOR ,
RUSSIAN_LANGUAGE_OPERATOR ,
SPANISH_LANGUAGE_OPERATOR ,
NATIONAL_NETWORKS_OPERATOR_SERVICE,
ORDINARY_CALLING_SUBSCRIBER,
CALLING_SUBSCRIBER_WITH_PRIORITY,
DATA_CALL,
TEST_CALL,
PAY_PHONE,
EMERGENCY_SERVICE_CALL_IN_PROGRESS,
HIGH_PRIORITY_CALL_INDICATION,
NSEP_CALL,
NETWORK_SPECIFIC_USE,
RESERVED
};

//
// Carrier Identification
//

enum CI_NETWORK_IDENTIFICATION_PLAN {
  UNKNOWN,
  THREE_DIGIT_CARRIER_IDENT_CODE,
  FOUR_DIGIT_CARRIER_IDENT_CODE,
  SPARE
};

enum CI_TYPE_OF_NETWORK_IDENTIFICATION {
  SPARE,
  NATIONAL_NETWORK_IDENTIFICATION
};

struct CARRIER_IDENTIFICATION {
  CI_NETWORK_IDENTIFICATION_PLAN networkIdentPlan;
  CI_TYPE_OF_NETWORK_IDENTIFICATION typeOfNetwork;
  string carrierID;
};

//
// Carrier Selection
//

enum CARRIER_SELECTION {
  NO_INDICATION,
  SUBS_DESIGNATED_PRESELECTED_CARRIER,
  SUBS_DESIGNATED_INPUT_CARRIER,
  SUBS_DESIGNATED_UNDETREMINED_CARRIER,
 DESIGNATED_BY_CALLER_CARRIER,
 SPARE,
 RESERVED
};

//
// Cause Indicator
//

enum CI_LOCATION {
 USER,
 LOCAL_PRIVATE_NETWORK,
 LOCAL_LOCAL_NETWORK,
 TRANSIT_NETWORK,
 REMOTE_LOCAL_NETWORK,
 REMOTE_PRIVATE_NETWORK,
 INTERNATIONAL_NETWORK,
 UNKNOWN,
 SPARE
};

enum CI_CODING_STANDARD {
 ITU_TS_STANDARD,
 RESERVED_FOR_INTL,
 ANSI_STANDARD,
 RESERVED
};

struct CAUSE_INDICATORS {
 sequence<CI_LOCATION> location;
 sequence<CI_CODING_STANDARD> codingStandard;
 boolean diagnosticsFlag;
 bytes causeValue;
 bytes diagnostics;
};

//
// Charge Number
//

enum CN_NATURE_OF_ADDR_IND {
 ANICallingSubNumber,
 ANICallingNotAvail,
 ANICallingNatNumber,
 ANICalledSubNumber,
 ANICalledNotAvail,
 ANICalledNatNumber
};

struct CHARGE_NUMBER {
  CN_NATURE_OF_ADDR_IND natureOfAddrInd;
  ODD_EVEN_BIT oddEvenBit;
  NUMBERING_PLAN numberingPlan;
  string addressSignal;
};

//
// Circuit Assignment Map
//

enum CIRCUIT_ASSIGNMENT_MAP_TYPE {
  DS1
};

enum CIRCUIT_ASSIGNMENT_MAP_STATUS {
    CIRCUIT_USED,
    CIRCUIT_NOT_USED
};

typedef sequence<CIRCUIT_ASSIGNMENT_MAP_STATUS>
CIRCUIT_ASSIGNMENT_MAP_STATUS_ARRAY;

struct CIRCUIT_ASSIGNMENT_MAP {
  CIRCUIT_ASSIGNMENT_MAP_TYPE type;
  CIRCUIT_ASSIGNMENT_MAP_STATUS_ARRAY status_array;
};

//
// Circuit Group Characterictics Indicator
//

enum CGCI_CARRIERINDICATOR {
  Unknown,
  Analog,
  Digital,
  DigitalAndAnalog
};

enum CGCI_DOUBLESEIZINGCTRLIND {
  NoCktCtrl,
  OddCIC,
  EvenCIC,
  AllCktCtrl
};

enum CGCI_ALARMCARRIERIND {
  Unknown,
  SoftwareHandling,
  HardwareHandling
};
```

```
enum CGCI_CONTINUITYCHKREQIND {
  Unknown,
  None,
  Statistical,
  PerCall
};

struct CKT_GRP_CHAR_INDICATORS {
  CGCI_CARRIERINDICATOR carrierIndicator;
  CGCI_DOUBLESEIZINGCTRLIND doubleSeizingCtrlInd;
  CGCI_ALARMCARRIERIND alarmCarrierInd;
  CGCI_CONTINUITYCHKREQIND continuityChkReqInd;
};

//
// Circuit Group Supervision Message Type Indicator
//

enum CGSMTI_BLOCKINGTYPEIND {
  WithoutRelease,
  WithImmediateRelease,
  RsvdForNationalUse
};


struct CKT_GRP_SUPERVISION_MSG_TYPE_IND {
  CGSMTI_BLOCKINGTYPEIND blockingTypeInd;
};

//
// Circuit Identification Name
//

struct CKT_IDENT_NAME {
  string trunkNumber;
  string CLLI_A;
  string CLLI_Z;
};

//
// Circuit State Indicator
//

enum CKT_STATE_IND {
  Transient,
  Unequiped,
  IncomingBusyActive,
  IncomingBusyLocallyBlocked,
  IncomingBusyRemotelyBlocked,
  IncomingBusyLocalAndRemoteBlocked,
  OutgoingBusyActive,
  OutgoingBusyLocallyBlocked,
  OutgoingBusyRemotelyBlocked,
  OutgoingBusyLocalAndRemoteBlocked,
  Idle,
  IdleLocallyBlocked,
  IdleRemotelyBlocked,
  IdleLocalAndRemoteBlocked
};

typedef sequence<CKT_STATE_IND> CKT_STATE_IND_ARRAY;

//
// Circuit Validation Response Indicator
//

enum CVRI_STATE {
  Successful,
  Failure
};

struct CKT_VALID_RESPONSE_IND {
  CVRI_STATE state;
};

//
// Common Language Location Indicator
//

struct CLLI_STRUCT {
  string town;
  string state;
  string building;
  string building_subdivision;
};

//
// Connection Request
//

struct CONNECTION_REQUEST{
  bytes localReference;
  bytes pointCode;
  octet protocolClass;
  octet credit;
};

//
// Continuity Indicators
```

```
//

enum CONTINUITY_INDICATORS {
  CONTINUITY_CHECK_FAILED,
  CONTINUITY_CHECK_SUCCESSFUL
};

//
// Event Information
//

enum EVENT_INDICATOR {
  SPARE ,
  ALERTING,
  PROGRESS,
  IN_BAND_INFO,
  CALL_FORWARDED_ON_BUSY,
  CALL_FORWARDED_ON_NO_REPLY,
  CALL_FORWARDED_UNCONDITIONAL,
  NOTIFICATION_FOR_SUPP_SRVC,
  SERVICE_INFO_INCLUDED,
  RESERVER
};

enum EVENT_PRESENTATION {
  NO_INDICATION,
  PRESENTATION_RESTRICTED
};

struct EVENT_INFORMATION {
  EVENT_INDICATOR eventIndicator;
  EVENT_PRESENTATION eventPresentation;
};

//
// Forward Call Indicators
//

enum INCOMING_INTERNATIONAL_CALL_INDICATOR {
  NOT_AN_INCOMING_INTERNATIONAL_CALL,
  INCOMING_INTERNATIONAL_CALL
};

enum ISDN_USER_PART_PREFERENCE_INDICATOR {
  ISUP_PREFERED_ALL_THE_WAY,
  ISUP_NOT_REQUIRED_ALL_THE_WAY,
  ISUP_REQUIRED_ALL_THE_WAY
};

enum PORTED_NUMBER_TRANSLATION_INDICATOR {
  NOT_TRANSLATED,
  TRANSLATED
 };

 struct FORWARD_CALL_INDICATORS {
   INCOMING_INTERNATIONAL_CALL_INDICATOR
incoming_International_Call_Indicator;
   END_TO_END_METHOD_INDICATOR
end_To_End_Method_Indicator;
   INTERWORKING_INDICATOR interworking_Indicator;
   IAM_SEGMENTATION_INDICATOR iam_Segmentation_Indicator;
   ISDN_USER_PART_INDICATOR isdn_User_Part_Indicator;
   ISDN_USER_PART_PREFERENCE_INDICATOR
isdn_User_Part_Preference_Indicator;
   ISDN_ACCESS_INDICATOR isdn_Access_Indicator;
   SCCP_METHOD_INDICATOR sccp_Method_Indicator;
   PORTED_NUMBER_TRANSLATION_INDICATOR
ported_Number_Translation_Indicator;
 };

 //
 // Generic Address
 //

 enum TYPE_OF_ADDRESS {
   DialedNumber,
   DestinationNbr,
   NetworkScreening,
   NotNetworkScreening,
   CompletionNumber,
   PortedNumber,
   AlternatelyBilledNumber,
   AssociatedForwardNumber,
   TransferNumber6,
   TransferNumber5,
   TransferNumber4,
   TransferNumber3,
   TransferNumber2,
   TransferNumber1,
   CESID
 };

 enum NATURE_OF_ADDRESS_INDICATOR_KIND {DIALED_DIGITS,
SUPPLEMENTAL, COMPLETION, PORTED};

 union NATURE_OF_ADDRESS_INDICATOR_UNION switch
(NATURE_OF_ADDRESS_INDICATOR_KIND) {
   case DIALED_DIGITS:
NATURE_OF_ADDRESS_INDICATOR_TYPE1 aDialedDigits;
```

```
  case SUPPLEMENTAL:
NATURE_OF_ADDRESS_INDICATOR_TYPE2 aSupplemental;
  case COMPLETION:   NATURE_OF_ADDRESS_INDICATOR_TYPE3
aCompletion;
  case PORTED:       NATURE_OF_ADDRESS_INDICATOR_TYPE4
aPorted;
 };

 struct GENERIC_ADDRESS {
  TYPE_OF_ADDRESS typeOfAddr;
  NATURE_OF_ADDRESS_INDICATOR_UNION natureOfAddr;
  ODD_EVEN_BIT oddEvenBit;
  NUMBERING_PLAN numberingPlan;
  PRESENTATION presentation;
  string addressSignal;
 };

 //
 // Generic Digits
 //

 enum TYPE_OF_DIGITS {
  ACCOUNT_CODE,
  AUTHORIZATION_CODE,
  PRIVATE_NETWORK_TRAVELLING_CLASS_MARK,
  CELL_SITE_SECTOR_IDENTIFIER,
  ORIGINATING_PARTY_SERVICE_PROVIDER,
  BILL_TO_NUMBER
 };

 enum ENCODING_SCHEME {
  BCD_EVEN,
  BCD_ODD,
  IA5,
  BINARY
 };

 struct GENERIC_DIGITS {
  TYPE_OF_DIGITS type_of_digits;
  ENCODING_SCHEME encoding_scheme;
  string digits;
 };

 //
 // Generic Name
 //

 enum AVAILABILITY {
  NAME_AVAILABLE,
  NAME_NOT_AVAILABLE
```

```
};

enum TYPE_OF_NAME {
  CALLING_NAME,
  ORIGINAL_CALLED_NAME,
  REDIRECTING_NAME,
  CONNECTED_NAME
};

enum GENERIC_NAME_PRESENTATION {
  PRESENTATION_ALLOWED,
  PRESENTATION_RESTRICTED,
  BLOCKING_TOGGLE,
  NO_INDICATION
};

struct GENERIC_NAME {
  GENERIC_NAME_PRESENTATION presentation;
  AVAILABILITY availability;
  TYPE_OF_NAME type_of_name;
  string name;
};

//
// Information Indicators
//

enum CALLING_PARTY_ADDRESS_RESPONSE_INDICATOR {
  NOT_INCLUDED,
  NOT_AVAILABLE,
  SPARE,
  INCLUDED_HOLD_NOT_PROVIDED
};

enum HOLD_PROVIDED_INDICATOR {
  NOT_PROVIDED,
  PROVIDED
};

enum CALLING_PARTY_CATEGORY_RESPONSE_INDICATOR {
  NOT_INCLUDED,
  INCLUDED
};

enum CHARGE_INFORMATION_RESPONSE_INDICATOR {
  NOT_INCLUDED,
  INCLUDED
};

enum SOLICITED_INFORMATION_INDICATOR {
   SOLICITED,
   UNSOLICITED
  };

  enum
MULTILOCATION_BUSINESS_GROUP_INFO_RESPONSE_INDICATO
R {
   NOT_INCLUDED,
   INCLUDED
  };

  struct INFORMATION_INDICATORS {
   CALLING_PARTY_ADDRESS_RESPONSE_INDICATOR
    calling_party_address_response_indicator;
   HOLD_PROVIDED_INDICATOR
    hold_provided_indicator;
   CALLING_PARTY_CATEGORY_RESPONSE_INDICATOR
    calling_party_category_response_indicator;
   CHARGE_INFORMATION_RESPONSE_INDICATOR
    charge_information_response_indicator;
   SOLICITED_INFORMATION_INDICATOR
    solicited_information_indicator;

MULTILOCATION_BUSINESS_GROUP_INFO_RESPONSE_INDICATO
R
    multilocation_business_group_info_response_indicator;
  };

  //
  // Information Request Indicator
  //

  enum CALLING_PARTY_ADDRESS_REQUEST_INDICATOR {
   NOT_REQUESTED,
   REQUESTED
  };

  enum INFORMATION_REQUEST_HOLDING_INDICATOR {
   NOT_REQUESTED,
   REQUESTED
  };

  enum CALLING_PARTY_CATEGORY_REQUEST_INDICATOR {
   NOT_REQUESTED,
   REQUESTED
  };

  enum CHARGE_INFORMATION_REQUEST_INDICATOR {
   NOT_REQUESTED,
   REQUESTED
```

```
};

enum MALICIOUS_CALL_ID_REQUEST_INDICATOR {
 NOT_REQUESTED,
 REQUESTED
};

enum MULTILOCATION_BUSINESS_GROUP_INFO_INDICATOR {
 NOT_REQUESTED,
 REQUESTED
};

struct INFORMATION_REQUEST_INDICATOR {
 CALLING_PARTY_ADDRESS_REQUEST_INDICATOR
  calling_party_address_request_indicator;
 INFORMATION_REQUEST_HOLDING_INDICATOR
  holding_indicator;
 CALLING_PARTY_CATEGORY_REQUEST_INDICATOR
  calling_party_category_request_indicator;
 CHARGE_INFORMATION_REQUEST_INDICATOR
  charge_information_request_indicator;
 MALICIOUS_CALL_ID_REQUEST_INDICATOR
  malicious_call_id_request_indicator;
 MULTILOCATION_BUSINESS_GROUP_INFO_INDICATOR
  multilocation_business_group_info_indicator;
};

//
// Jurisdiction Information
//

struct JURISDICTION_INFORMATION {
 string addressSignal;
};

//
// Nature of Connection Indicator
//

enum SATELLITE_INDICATOR {
 NO_SATELLITE_CIRCUIT,
 ONE_SATELLITE_CIRCUIT,
 TWO_SATELLITE_CIRCUIT,
 THREE_OR_MORE_SATELLITE_CIRCUIT
};

enum CONTINUITY_CHECK_INDICATOR {
 CONTINUITY_CHECK_NOT_REQUIRED,
 CONTINUITY_CHECK_REQUIRED,
 CONTINUITY_CHECK_ON_PREVIOUS_CIRCUIT,
  SPARE
 };

 struct NATURE_OF_CONNECTION_INDICATOR {
  SATELLITE_INDICATOR satellite_indicator;
  CONTINUITY_CHECK_INDICATOR continuity_check_indicator;
  ECHO_CONTROL_DEVICE_INDICATOR
echo_control_device_indicator;
 };

 //
 // Network Management Control
 //

 enum TEMPORARY_ALTERNATIVE_ROUTING {
  TAR_NO_INDICATION,
  TAR_CONTROLLED_CALL
 };

 struct NETWORK_MANAGEMENT_CONTROLS {
  TEMPORARY_ALTERNATIVE_ROUTING
temporaryAlternativeRouting;
 };

 typedef sequence<NETWORK_MANAGEMENT_CONTROLS>
NETWORK_MANAGEMENT_CONTROLS_ARRAY;

 //
 // Network Transport Parameter
 //

 struct NETWORK_TRANSPORT_PARAMETER {
  sequence<string> parmNames;
  sequence<bytes> parmValues;
  bytes networkTransport;
 };

 //
 // Notification Indicator
 //

 enum NOTIFICATION_IND {
  CALL_COMPLETION_DELAY,
  CALL_IS_A_WAITING_CALL,
  TRANSFER_IN_PROGRESS,
  ISOLATED_FROM_CONFERENCE_CALL,
  SPLIT_FROM_CONFERENCE_CALL,
  REATTACHED_TO_CONFERENCE_CALL,
  ADDED_TO_CONFERENCE_CALL,
  REMOTE_HOLD,
  REMOTE_HOLD_RELEASED,
  CALL_IS_FORWARDED,
  SPARE,
  RESERVED
 };

 struct NOTIFICATION_INDICATOR {
  NOTIFICATION_IND notification_ind;
 };

 typedef sequence<NOTIFICATION_INDICATOR>
NOTIFICATION_INDICATOR_ARRAY;

 //
 // Operator Service Information
 //

 enum INFORMATION_TYPE {
  UNKNOWN,
  ORIGINAL_ACCESS_PREFIX,
  BILL_TO_INFO_ENTRY_TYPE_AND_HANDLE_TYPE,
  BILL_TO_TYPE,
  BILL_TO_SPECIFIC_INFO,
  SPECIAL_HANDLING,
  ACCESS_SIGNALING
 };

 enum INFOMATION_VALUE_001 {
  UNKNOWN,
  A_1OR011,
  A_0OR01,
  A_0
 };

 enum INFOMATION_VALUE_010 {
  UNKNOWN_UNKNOW_HANDLING,
  OPERATOR_STATION_HANDLING,
  OPERATOR_PERSON_HANDLING,
  TONE_INPUT_STATION_HANDLING,
  UNKNOWN_STATION_HANDLING,
  UNKNOWN_PERSON_HANDLING,
  OPERATOR_UNKNOWN_HANDLING,
  TONE_INPUT_UNKNOWN_HANDLING,
  TONE_INPUT_PERSON_HANDLING,
  SPOKEN_INPUT_UNKNOWN_HANDLING,
  SPOKEN_INPUT_STATION_HANDLING,
  SPOKEN_INPUT_PERSON_HANDLING
 };

 enum INFOMATION_VALUE_011 {
  UNKNOWN,
  CARD14DIGIT,
  CARD89C,
  CARDOTHER,
  COLLECT,
  THIRDNUMBER,
  SENTPAID
 };

 enum INFOMATION_VALUE_100 {
  UNKNOWN,
  NIDB_AUTHORIZE,
  NIDB_REPORT_VERIFY_AUTOMATED,
  NIDB_REPORT_VERIFY_OPERATOR,
  NO_NIDB_QUERY,
  NO_NIDB_RESPONSE,
  NIDB_REPORT_UNAVAILABLE,
  NO_NIDB_RESPONSE_TIMEOUT,
  NO_NIDB_RESPONSE_REJECT,
  NO_NIDB_RESPONSE_ACG,
  NO_NIDB_RESPONSE_SCCP_FAIL
 };

 enum INFOMATION_VALUE_101 {
  UNKNOWN,
  CALL_COMPLETION,
  RATE_INFO,
  TROUBLE_REPORT,
  TIME_CHARGE,
  CREDIT_REPORT,
  GENERAL_ASSIST
 };

 enum INFOMATION_VALUE_111 {
  UNKNOWN,
  DIAL_PULSE,
  DIAL_TONE
 };

 enum INFORMATION_VALUE_KIND {kind_001, kind_010, kind_011,
kind_100, kind_101, kind_111};

 union INFORMATION_VALUE_UNION switch
(INFORMATION_VALUE_KIND) {
  case kind_001: INFOMATION_VALUE_001 a001;
  case kind_010: INFOMATION_VALUE_010 a010;
  case kind_011: INFOMATION_VALUE_011 a011;
  case kind_100: INFOMATION_VALUE_100 a100;
  case kind_101: INFOMATION_VALUE_101 a101;
  case kind_111: INFOMATION_VALUE_111 a111;
```

```
};

struct OPERATOR_SERVICE_INFO {
  INFORMATION_TYPE informationType;
  INFORMATION_VALUE_UNION informationValue;
};

typedef sequence<OPERATOR_SERVICE_INFO>
OPERATOR_SERVICE_INFO_ARRAY;

//
// Optional Backward Call Indicator
//

enum IN_BAND_INFORMATION_INDICATOR {
  NO_INDICATION,
  IN_BAND_INFO_OR_A_PATTERN_IS_AVAIL
};

enum CALL_FORWARDING_MAY_OCCUR_INDICATOR {
  NO_INDICATION,
  MAY_OCCUR
};

enum NATIONAL_USE {
  NATIONAL
};

enum NETWORK_EXCESSIVE_DELAY_INDICATOR {
  NO_INDICATION,
  NETWORK_EXCESSIVE_DELAY_ENCOUNTERED
};

enum USER_NETWORK_INTERACTION_OCCURS {
  NO_INDICATION,
  CUT_THROUGH_IN_BOTH_DIR
};

struct OPTIONAL_BACKWARD_CALL_INDICATORS {
  IN_BAND_INFORMATION_INDICATOR
    in_band_information_indicator;
  CALL_FORWARDING_MAY_OCCUR_INDICATOR
    call_forwarding_may_occur_indicator;
  NATIONAL_USE
    national_use;
  NETWORK_EXCESSIVE_DELAY_INDICATOR
    network_excessive_delay_indicator;
  USER_NETWORK_INTERACTION_OCCURS
    user_network_interaction_occurs;
};
```

```
//
// Originating Call Number
//

struct ORIGINAL_CALLED_NUMBER {
  NATURE_OF_ADDRESS_INDICATOR_TYPE2 addressNatureInd;
  ODD_EVEN_BIT oddEvenBit;
  PRESENTATION presentation;
  NUMBERING_PLAN numberingPlan;
  string addressSignal;
};

//
// Originating Line Information
//

typedef octet BINARY_EQUIVALENT_OF_THE_II_DIGITS;

struct ORIGINATING_LINE_INFORMATION
{
  BINARY_EQUIVALENT_OF_THE_II_DIGITS
binary_equivalent_of_the_two_digits;
};

//
// Outgoing Trunk Number
//

enum MULTI_LVL_PP {
   DEFENSE_SWITCH_NETWORK,
   SPARE
};

struct OUTGOING_TRUNK_GROUP_NUMBER {
   long outgoing_trunk_group_number;
};

//
// Precedence Level
//

enum PRECEDENCE_LEVEL {
  FLASH_OVERRIDE,
  FLASH,
  IMMEDIATE,
  PRIORITY,
  ROUTINE,
  SPARE
};
```

```
enum LOOK_AHEAD_FOR_BUSY {
 LOOK_AHEAD_FOR_BUSY_ALLOWED,
 LOOK_AHEAD_FOR_BUSY_NOT_ALLOWED,
 PATH_RESERVED,
 RESERVED
};

struct PRECEDENCE {
 PRECEDENCE_LEVEL precedence_level;
 LOOK_AHEAD_FOR_BUSY look_ahead_for_busy;
 long network_identity;
 long MLPP_service_domain;
};

//
// Range and Status
//

enum STATUS {
 NO_BLOCKING,
 BLOCKING,
 BLOCKED,
 UNBLOCKED,
 NO_BLOCKING_ACKNOWLEDGMENT,
 BLOCKING_ACKNOWLEDGMENT,
 NO_UNBLOCKING,
 UNBLOCKING,
 NO_UNBLOCKING_ACKNOWLEDGMENT,
 UNBLOCKING_ACKNOWLEDGMENT
};

typedef sequence<STATUS> STATUS_ARRAY;
typedef short RANGE;

struct RANGE_AND_STATUS {
 RANGE range;
 STATUS_ARRAY status_array;
};

//
// Redirect Capability
//

enum REDIRECT_CAPABILITY_ENUM {
 REDIRECTION_POSSIBLE_BEFORE_ACM,
 REDIRECTION_POSSIBLE_BEFORE_ANM,
 REDIRECTION_POSSIBLE_ANYTIME
};
```

```
struct REDIRECT_CAPABILITY {
  REDIRECT_CAPABILITY_ENUM redirectCapability;
};

typedef sequence<REDIRECT_CAPABILITY>
REDIRECT_CAPABILITY_ARRAY;

//
// Redirect Counter
//

struct REDIRECT_COUNTER {
  octet redirectCounter;
};

//
// Redirecting Number
//

struct REDIRECTING_NUMBER {
  NATURE_OF_ADDRESS_INDICATOR_TYPE2 addressNatureInd;
  ODD_EVEN_BIT oddEvenBit;
  PRESENTATION presentation;
  NUMBERING_PLAN numberingPlan;
  string addressSignal;
};

//
// Redirection Information
//

enum ORIGINAL_REDIRECTING_REASON {
  UNKNOWN_NOT_AVAILABLE,
  USER_BUSY,
  NO_REPLY,
  UNCONDITIONAL,
  SPARE,
  RESERVED
};

enum REDIRECTION_COUNTER {
  NO_REDIRECTION_HAS_OCCURED,
  REDIRECTED_1_TIME,
  REDIRECTED_2_TIMES,
  REDIRECTED_3_TIMES,
  REDIRECTED_4_TIMES,
  REDIRECTED_5_TIMES,
  REDIRECTED_6_TIMES,
  REDIRECTED_7_TIMES,
  REDIRECTED_8_TIMES,
 REDIRECTED_9_TIMES,
 REDIRECTED_10_TIMES,
 REDIRECTED_11_TIMES,
 REDIRECTED_12_TIMES,
 REDIRECTED_13_TIMES,
 REDIRECTED_14_TIMES,
 REDIRECTED_15_TIMES
};

enum REDIRECTING_REASON {
 UNKNOWN_NOT_AVAILABLE,
 USER_BUSY,
 NO_REPLY,
 UNCONDITIONAL,
 SPARE
};

struct REDIRECTION_INFORMATION {
 REDIRECTION_COUNTER redirection_Counter;
 ORIGINAL_REDIRECTING_REASON original_redirecting_reason;
 REDIRECTING_REASON redirecting_reason;
};

//
// Redirection Number
//

struct REDIRECTION_NUMBER {
 NATURE_OF_ADDRESS_INDICATOR_TYPE3 addressNatureInd;
 ODD_EVEN_BIT oddEvenBit;
 NUMBERING_PLAN numberingPlan;
 string addressSignal;
};

//
// Remote Operation
//

enum PROTOCOL_PROFILE {
 SPARE,
 REMOTE_OPERATION_PROTOCOL
};

struct REMOTE_OPERATIONS {
 PROTOCOL_PROFILE protocol_profile;
 bytes components;
};

//
// Service Activation
```

```
//

enum SERVICE_ACTIVATION_ENUM {
  RESERVED_INTERNATIONAL,
  CALL_WAITING_ORIGINATING_INVOKED,
  DIAL_CALL_WAITING_INVOKED,
  COMPLETE_CALL_REQUEST_ISUP_USED_ALL_THE_WAY,
  COMPLETE_CALL_REQUEST_ISUP_NOT_USED_ALL_THE_WAY,
  SPARE,
  RESERVED_NETWORK_SPECIFIC
};

struct SERVICE_ACTIVATION {
  SERVICE_ACTIVATION_ENUM service_activation_enum;
};

typedef sequence<SERVICE_ACTIVATION>
SERVICE_ACTIVATION_ARRAY;

//
// Service Code
//

struct SERVICE_CODE {
  long service_code;
};

//
// Special Processing Request
//

enum SPECIAL_PROCESSING_REQUEST_ENUM {
  SPARE,
  SERVICE_PROCESSING_REQUIRED,
  RESERVED_FOR_INTERNATIONAL_USE,
  RESERVED_FOR_NATIONAL_USE,
  RESERVED_FOR_NETWORK_SPECIFIC_USE
};

typedef octet RESERVED_FOR_INTERNATIONAL_USE;
typedef octet RESERVED_FOR_NATIONAL_USE;
typedef octet RESERVED_FOR_NETWORK_SPECIFIC_USE;

struct SPECIAL_PROCESSING_REQUEST {
  SPECIAL_PROCESSING_REQUEST_ENUM special_processing_rq;
  RESERVED_FOR_INTERNATIONAL_USE
reserved_for_international_use;
  RESERVED_FOR_NATIONAL_USE reserved_for_national_use;
  RESERVED_FOR_NETWORK_SPECIFIC_USE
reserved_for_network_specific_use;
```

```
};

//
// Suspend Resume Indicator
//

enum SUSPEND_RESUME_INDICATOR {
  ISDN_SUBSCRIBER_INITIATED,
  NETWORK_INITIATED
};

//
// Transaction Request
//

struct TRANSACTION_REQUEST {
  bytes transactionID;
  bytes SCCP_Address;
};

//
// Transit Network Selection
//

enum
NETWORK_IDENTIFICATION_PLAN_NATIONAL_ANSI_NETWORKS {
   UNKNOWN,
   THREE_DIGIT_CARRIER_IDENTIFICATION_WITH_CIRCUIT_CODE,
   FOUR_DIGIT_CARRIER_IDENTIFICATION_WITH_CIRCUIT_CODE,
   RESERVED,
   RESERVED_FOR_NETWORK_SPECIFIC_USE_FLAG
};

enum
NETWORK_IDENTIFICATION_PLAN_INTERNATIONAL_NETWORKS {
   UNKNOWN,
   PUBLIC_DATA_NETWORK_IDENTIFICATION_CODE,
   PUBLIC_LAND_MOBILE_NETWORK_ID_CODE
};

enum TYPE_OF_NETWORK_IDENTIFICATION {
  ITU_STANDARDIZED_IDENTIFICATION,
  NATIONAL_NETWORK_IDENTIFICATION
};

enum TRANSIT_NETWORK_SELECTION_DIGIT {
  DIGIT0,
  DIGIT1,
  DIGIT2,
  DIGIT3,
  DIGIT4,
  DIGIT5,
  DIGIT6,
  DIGIT7,
  DIGIT8,
  DIGIT9,
  SPARE,
  CODE11,
  CODE12,
  END_OF_PULSE_SIGNAL
};

enum CIRCUIT_CODE {
  UNSPECIFIED,
  INTERNATIONAL_CALL_NO_OPERATOR_REQUESTED,
  INTERNATIONAL_CALL_OPERATOR_REQUESTED,
  SPARE,
  RESERVED_FOR_NETWORK_SPECIFIC_USE_FLAG
};

struct TRANSIT_NETWORK_SELECTION {
  NETWORK_IDENTIFICATION_PLAN_NATIONAL_ANSI_NETWORKS
network_identification_plan_ansi_networks;

NETWORK_IDENTIFICATION_PLAN_INTERNATIONAL_NETWORKS
network_identification_plan_international_networks;
  TYPE_OF_NETWORK_IDENTIFICATION
type_of_network_identification;
  TRANSIT_NETWORK_SELECTION_DIGIT digit_one;
  TRANSIT_NETWORK_SELECTION_DIGIT digit_two;
  TRANSIT_NETWORK_SELECTION_DIGIT digit_three;
  TRANSIT_NETWORK_SELECTION_DIGIT digit_four;
  CIRCUIT_CODE circuit_code;
  octet circuit_code_network_specific_use;
  octet ansi_network_specific_use;
};

//
// Transmission Medium
//

enum TRANSMISSION_MEDIUM_USED_VALUE {
  SPEECH,
  RESERVED_64KBPS_UNRESTRICTED,
  AUDIO31KHZ,
  RESERVED_64KBPS_PREFERRED
};

struct TRANSMISSION_MEDIUM_USED {
  TRANSMISSION_MEDIUM_USED_VALUE
transmission_medium_used_value;
 };

 //
 // User Service Information
 // User Service Information Prime
 //

 enum INFO_TRANSFER_CAPABILITY {
  Speech,
  UnrestrictedDigital,
  RestrictedDigital,
  Audio3100Hz,
  Audio7kHz
 };

 enum INFO_CODINGSTANDARD {
  ITUStandard,
  NationalStandard
 };

 enum INFO_TRANSFER_RATE {
  codeforPacketMode,
  kbps64,
  kbps384,
  kbps1472,
  kbps1536,
  kbps1920,
  Multirate
 };

 enum INFO_TRANSFER_MODE {
  circuit,
  packet
 };

 enum USER_INFO_ESTABLISHMENT {
  Demand
 };

 enum USER_INFO_CONFIGURATION {
  PointtoPoint
 };

 enum USER_INFO_STRUCTURE {
  Default,
  Integrity8kHz,
  ServiceDataUnitIntegrity,
  Unstructured
```

```
};

enum USER_INFO_SYMMETRY {
  Bidirectional
};

enum USER_INFO_LAYER1PROTOCOL {
  NotPresent,
  ITUStandardRateAdaptionV110,
  G771ulawSpeech,
  G722andG725Audio,
  NONITUStandardRateAdaption,
  ITUStandardRateAdaptionV120,
  ITUStandardRateAdaptionX31HDLC
};

enum USER_INFO_LAYER2PROTOCOL {
  NotPresent,
  I144OrQ921,
  X25LinkLevel
};

enum USER_INFO_LAYER3PROTOCOL {
  NotPresent,
  ANSIT1607,
  X25Packet
};

struct USER_SERVICE_INFORMATION {
  INFO_TRANSFER_CAPABILITY transferCapability;
  INFO_CODINGSTANDARD codingStandard;
  INFO_TRANSFER_RATE transferRate;
  INFO_TRANSFER_MODE transferMode;
  USER_INFO_ESTABLISHMENT establishment;
  USER_INFO_CONFIGURATION configuration;
  USER_INFO_STRUCTURE structure;
  INFO_TRANSFER_RATE destToOriginationTransferRate;
  USER_INFO_SYMMETRY symmetry;
  octet multirateRateMultiple;
  USER_INFO_LAYER1PROTOCOL userlayer1protocol;
  USER_INFO_LAYER2PROTOCOL userlayer2protocol;
  USER_INFO_LAYER3PROTOCOL userlayer3protocol;
};

struct USER_SERVICE_INFORMATION_PRIME {
  INFO_TRANSFER_CAPABILITY transferCapability;
  INFO_CODINGSTANDARD codingStandard;
  INFO_TRANSFER_RATE transferRate;
  INFO_TRANSFER_MODE transferMode;
  USER_INFO_ESTABLISHMENT establishment;
  USER_INFO_CONFIGURATION configuration;
  USER_INFO_STRUCTURE structure;
  INFO_TRANSFER_RATE destToOriginationTransferRate;
  USER_INFO_SYMMETRY symmetry;
  octet multirateRateMultiple;
  USER_INFO_LAYER1PROTOCOL userlayer1protocol;
  USER_INFO_LAYER2PROTOCOL userlayer2protocol;
  USER_INFO_LAYER3PROTOCOL userlayer3protocol;
};

//
// User to User Indicator
//

enum USER_TO_USER_INDICATOR_TYPE {
  REQUEST,
  RESPONSE
};

enum USER_TO_USER_INDICATOR_RESPONSE {
  NONE,
  SERVICE
};

enum NETWORK_DISCARD_INDICATOR {
  NO_INFORMATION,
  USER_TO_USER_INFORMATION_DISCARDED_BY_NETWORK
};

struct USER_TO_USER_INDICATOR {
  USER_TO_USER_INDICATOR_TYPE user_to_user_indicator;
  USER_TO_USER_INDICATOR_RESPONSE
user_to_user_indicator_service1;
  USER_TO_USER_INDICATOR_RESPONSE
user_to_user_indicator_service2;
  USER_TO_USER_INDICATOR_RESPONSE
user_to_user_indicator_service3;
  NETWORK_DISCARD_INDICATOR network_discard_indicator;
};

//
// User to User Information
//

enum PROTOCOL_DISCRIMINATOR {
  USER_SPECIFIC,
  OSI_HIGH_LAYER,
  X244,
  RESERVED1,
  ASCII,
  X208X209,
  V120,
  T1607,
  RESERVED2,
  NATIONAL,
  RESERVED3
};

struct USER_TO_USER_INFORMATION {
  PROTOCOL_DISCRIMINATOR protocolDiscriminator;
  octet protocolDiscriminatorReserved2;
  octet protocolDiscriminatorReserved3;
  octet protocolDiscriminatorNational;
  bytes user_to_user_info;
};

//
//
// Beginning of ISUP Message Definition Section
//
//

//
// Address Complete
//

struct ACMMessage {
  // Mandatory Fixed Part
  BACKWARD_CALL_INDICATOR backwardCallIndicators;
  // Optional Parameters
  ACCESS_TRANSPORT accessTransport;
  BUSINESS_GROUP businessGroup;
  CALL_REFERENCE callReference;
  CAUSE_INDICATORS causeIndicators;
  CONNECTION_REQUEST connectionRequest;
  INFORMATION_INDICATORS informationIndicators;
  NETWORK_TRANSPORT_PARAMETER networkTransportParameter;
  NOTIFICATION_INDICATOR_ARRAY notificationIndicatorArray;
  OPTIONAL_BACKWARD_CALL_INDICATORS
optionalBackwardCallIndicators;
  REDIRECTION_INFORMATION redirectionInformation;
  REMOTE_OPERATIONS remoteOperations;
  SERVICE_ACTIVATION_ARRAY serviceActivationArray;
  TRANSMISSION_MEDIUM_USED TransmissionMediumUsed;
  USER_TO_USER_INDICATOR user_to_userIndicator;
  USER_TO_USER_INFORMATION user_to_userInformation;
};

//
// Answer
```

```
//

struct ANMMessage {
  // Optional Parameters
  ACCESS_TRANSPORT accessTransport;
  BACKWARD_CALL_INDICATOR backwardCallIndicators;
  BUSINESS_GROUP businessGroup;
  CALL_REFERENCE callReference;
  CONNECTION_REQUEST connectionRequest;
  INFORMATION_INDICATORS informationIndicators;
  NETWORK_TRANSPORT_PARAMETER networkTransportParameter;
  NOTIFICATION_INDICATOR_ARRAY notificationIndicatorArray;
  OPTIONAL_BACKWARD_CALL_INDICATORS
optionalBackwardCallIndicators;
  REMOTE_OPERATIONS remoteOperations;
  SERVICE_ACTIVATION_ARRAY serviceActivationArray;
  TRANSMISSION_MEDIUM_USED TransmissionMediumUsed;
  USER_TO_USER_INDICATOR user_to_userIndicators;
  USER_TO_USER_INFORMATION user_to_userInformation;
};

//
// Blocking
//
// struct BLOMessage {}
//

//
// Blocking Acknowledgement
//
// struct BLAMessage {}
//

//
// Call Progress
//

struct CPGMessage {
  // Mandatory Fixed Part
  EVENT_INFORMATION eventInformation;
  // Optional Parameters
  ACCESS_TRANSPORT accessTransport;
  BACKWARD_CALL_INDICATOR backwardCallIndicators;
  BUSINESS_GROUP businessGroup;
  CALL_REFERENCE callReference;
  CAUSE_INDICATORS causeIndicators;
  INFORMATION_INDICATORS informationIndicators;
  NETWORK_TRANSPORT_PARAMETER networkTransportParameter;
  NOTIFICATION_INDICATOR_ARRAY notificationIndicatorArray;
  OPTIONAL_BACKWARD_CALL_INDICATORS
optionalBackwardCallIndicators;
  REDIRECTION_NUMBER redirectionNumber;
  REMOTE_OPERATIONS remoteOperations;
  SERVICE_ACTIVATION_ARRAY serviceActivationArray;
  TRANSMISSION_MEDIUM_USED TransmissionMediumUsed;
  USER_TO_USER_INDICATOR user_to_userIndicator;
  USER_TO_USER_INFORMATION user_to_userInformation;
 };

 //
 // Circuit Group Blocking
 //

 struct CGBMessage {
  // Mandatory Fixed Part
  CKT_GRP_SUPERVISION_MSG_TYPE_IND
circuitGroupSupervisionMessageTypeIndicator;
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
 };

 //
 // Circuit Group Blocking Acknowledgement
 //

 struct CGBAMessage {
  // Mandatory Fixed Part
  CKT_GRP_SUPERVISION_MSG_TYPE_IND
circuitGroupSupervisionMessageTypeIndicator;
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
 };

 //
 // Circuit Group Reset
 //

 struct GRSMessage {
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
  // Optional Parameters
  CIRCUIT_ASSIGNMENT_MAP circuitAssignmentMap;
 };

 //
 // Circuit Group Reset Acknowledgement
 //

 struct GRAMessage {
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
  // Optional Parameters
  CIRCUIT_ASSIGNMENT_MAP circuitAssignmentMap;
};

//
// Circuit Group Unblocking
//

struct CGUMessage {
  // Mandatory Fixed Part
  CKT_GRP_SUPERVISION_MSG_TYPE_IND
circuitGroupSupervisionMessageTypeIndicator;
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
};

//
// Circuit Group Unblocking Acknowledgement
//

struct CGUAMessage {
  // Mandatory Fixed Part
  CKT_GRP_SUPERVISION_MSG_TYPE_IND
circuitGroupSupervisionMessageTypeIndicator;
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
};

//
// Circuit Query
//

struct CQMMessage {
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
  // Optional Parameters
  CIRCUIT_ASSIGNMENT_MAP circuitAssignmentMap;
};

//
// Circuit Query Response
//

struct CQRMessage {
  // Mandatory Variable Part
  RANGE_AND_STATUS rangeAndStatus;
  CKT_STATE_IND_ARRAY circuitStateIndicatorArray;
};
```

```
//
// Circuit Reservation
//

struct CRMMessage {
  // Mandatory Fixed part
  NATURE_OF_CONNECTION_INDICATOR
natureOfConnectionIndicators;
};

//
// Circuit Reservation Acknowledgement
//
// struct CRAMessage {}
//

//
// Circuit Validation Response
//

struct CVRMessage {
  // Mandatory Fixed Part
  CKT_VALID_RESPONSE_IND circuitValidationResponseIndicator;
  CKT_GRP_CHAR_INDICATORS circuitGroupCharacteristicIndicators;
  // Optional Parameters
  CKT_IDENT_NAME circuitIdentificationName;
  CLLI_STRUCT CLLICode;
};

//
// Circuit Validation Test
//
// struct CVTMessage {}
//

//
// Confusion
//

struct CFNMessage {
  // Mandatory Variable Part
  CAUSE_INDICATORS causeIndicators;
};

//
// Continuity
//

struct COTMessage {
  // Mandatory Fixed Part
  CONTINUITY_INDICATORS continuityIndicators;
};

//
// Continuity Check Request
//
// struct CCRMessage {}
//

//
// Exit
//

struct EXMMessage {
  // Optional Parameters
  OUTGOING_TRUNK_GROUP_NUMBER
outgoingTrunkGroupNumber;
};

//
// Facility
//

struct FACMessage {
  // Optional Parameters
  REMOTE_OPERATIONS remoteOperations;
  SERVICE_ACTIVATION_ARRAY serviceActivationArray;
};

//
// Forward Transfer
//

struct FOTMessage {
  // Optional Parameters
  CALL_REFERENCE callReference;
};

//
// Information
//

struct INFMessage {
  // Mandatory Fixed Part
  INFORMATION_INDICATORS informationIndicators;
  // Optional Parameters
  ACCESS_TRANSPORT accessTransport;
  BUSINESS_GROUP businessGroup;
  CALL_REFERENCE callReference;
```

```
    CALLING_PARTY_NUMBER callingPartyNumber;
    CALLING_PARTY_CATEGORY callingPartyCategory;
    CHARGE_NUMBER chargeNumber;
    CONNECTION_REQUEST connectionRequest;
    ORIGINATING_LINE_INFORMATION originatingLineInformation;
    REDIRECTING_NUMBER redirectingNumber;
    REDIRECTION_INFORMATION redirectionInformation;
    USER_TO_USER_INFORMATION user_to_userInformation;
  };

  //
  // Information Request
  //

  struct INRMessage {
    // Mandatory Fixed Part
    INFORMATION_REQUEST_INDICATOR
informationRequestIndicators;
    // Optional Parameters
    CALL_REFERENCE callReference;
    CONNECTION_REQUEST connectionRequest;
    NETWORK_TRANSPORT_PARAMETER networkTransportParameter;
  };

  //
  // Initial Address
  //

  struct IAMMessage {
    // Mandatory Fixed Part
    NATURE_OF_CONNECTION_INDICATOR
natureOfConnectionIndicators;
    FORWARD_CALL_INDICATORS forwardCallIndicators;
    CALLING_PARTY_CATEGORY callingPartyCategory;
    // Mandatory Variable Part
    USER_SERVICE_INFORMATION userServiceInformation;
    CALLED_PARTY_NUMBER calledPartyNumber;
    // Optional Parameters
    ACCESS_TRANSPORT accessTransport;
    BUSINESS_GROUP businessGroup;
    CALL_REFERENCE callReference;
    CALLING_PARTY_NUMBER callingPartyNumber;
    CARRIER_IDENTIFICATION carrierIdentification;
    CARRIER_SELECTION carrierSelection;
    CHARGE_NUMBER chargeNumber;
    CIRCUIT_ASSIGNMENT_MAP circuitAssignmentMap;
    CONNECTION_REQUEST connectionRequest;
    bytes egressService;
    GENERIC_ADDRESS genericAddress;
    GENERIC_DIGITS genericDigits;
  GENERIC_NAME genericName;
  octet hopCounter;
  INFORMATION_REQUEST_INDICATOR
informationRequestIndicators;
  JURISDICTION_INFORMATION jurisdictionInformation;
  NETWORK_MANAGEMENT_CONTROLS_ARRAY
networkManagementControlsArray;
  NETWORK_TRANSPORT_PARAMETER networkTransportParameter;
  OPERATOR_SERVICE_INFO_ARRAY operatorServiceInfoArray;
  ORIGINAL_CALLED_NUMBER originalCalledNumber;
  ORIGINATING_LINE_INFORMATION originatingLineInformation;
  PRECEDENCE precedence;
  REDIRECT_CAPABILITY_ARRAY redirectCapabilityArray;
  REDIRECT_COUNTER redirectCounter;
  REDIRECTING_NUMBER redirectingNumber;
  REDIRECTION_INFORMATION redirectionInformation;
  REMOTE_OPERATIONS remoteOperations;
  SERVICE_ACTIVATION_ARRAY serviceActivationArray;
  SERVICE_CODE serviceCode;
  SPECIAL_PROCESSING_REQUEST specialProcessingRequest;
  TRANSACTION_REQUEST transactionRequest;
  TRANSIT_NETWORK_SELECTION transitNetworkSelection;
  USER_SERVICE_INFORMATION_PRIME
userServiceInformationPrime;
  USER_TO_USER_INFORMATION user_to_userInformation;
 };

 //
 // Loop Back Acknowledgement
 //
 // struct LPAMessage {}
 //

 //
 // Pass Along
 //
 // struct PAMMessage {}
 //

 //
 // Release
 //

 struct RELMessage {
  // Mandatory Variable Part
  CAUSE_INDICATORS causeIndicators;
  // Optional Parameters
  ACCESS_TRANSPORT accessTransport;
  AUTOMATIC_CONGESTION_LEVEL automaticCongestionLevel;
  CALL_REFERENCE callReference;
 CHARGE_NUMBER chargeNumber;
 GENERIC_ADDRESS genericAddress;
 SERVICE_ACTIVATION_ARRAY serviceActivationArray;
 USER_TO_USER_INFORMATION user_to_userInformation;
};

//
// Release Complete
//
// struct RLCMessage {}
//

//
// Reset Circuit
//
// struct RSCMessage {}
//

//
// Resume
//

struct RESMessage {
 // Mandatory Fixed Part
 SUSPEND_RESUME_INDICATOR suspend_resumeIndicators;
 // Optional Parameters
 CALL_REFERENCE callReference;
};

//
// Suspend
//

struct SUSMessage {
 // Mandatory Fixed Part
 SUSPEND_RESUME_INDICATOR suspend_resumeIndicators;
 // Optional Parameters
 CALL_REFERENCE callReference;
};

//
// Unblocking
//
// struct UBLMessage {}
//

//
// Unblocking Acknowledgement
//
// struct UBAMessage {}
```

```
    //

    //
    // Unequipped Circuit Identification Code
    //
    // struct UCICMessage {}
    //
   };
```

Appendix B

```
#   Call Model for SGCP
#
#===============================================================
==
#   Events:
#
#           Format:   Event <eventName> <side-of-call>
#
#       SetTopBox Events
#
#           OffHook             - start call (Ingress) or answer ring (Egress)
#           DialComplete        - digits collected for called number (Ingress)
#           OnHook              - hang up (ingress or Egress)
#
#       Ingress Events
#
#           Created             - confirming Create
#           Answered            - showing answered call
#           Released            - confirming delete
#           Suspended           - Egress hung up
#           Resumed             - Egress picks back up during Suspend
#           Delete         - tair down local connection
#           TimerExpired- timer expired
#           Busy                - Egress indicates busy called number
#           Announcement        - an announcement must be identified
#           InvalidEndpoint     - Egress indicates invalid called number
#
#       Egress Events
#
#           Create              - establish connection
#           Released            - confirming delete
#           Delete         - tair down local connecting
#           TimerExpired- timer expired
#           Busy                - called number found busy on create
#           Announcement        - announcement must be played
#           InvalidEndpoint     - received invalid called number
#
#   Transitions:
#
#           Format:   Transition <curState> <nextState>
#
#           Idle           - connection inactive
#           Dialing        - Ingress is collecting digits for called number
#           Ringing        - Egress is ringing called number
#           Active - Connection is made, full duplex
#           Suspended      - Active session suspended
#           Releasing      - Ingress or Egress releasing local resources
#           Initiated      - Ingress waiting for create confirmation
#           Delivered     - Ingress waiting for Egress to answer
#           * (Any)       - wild card on any state
#
#         "wild card" matches on Transition State supported
#         curState=*  nextState=*
#           where * in curState means match on any current state
#               * in nextState means DONT change state
#
#   Actions:
#
#           Format:  Action <actionName> <parms>
#
#           CreateConnection   - Call Voip Gateway to create new connection
#           ModifyConnection   - Call Voip Gateway to Modify existing connection
#           DeleteConnection   - Call Voip Gateway to Delete existing connection
#           NotifyRequest      - Call Voip Gateway to be notified of events
#
#           AccountingStart    - Call Accounting Gateway Interface with Start
Record
#           AccountingStop     - Call Accounting Gateway Interface with Stop
Record
#
#           McapCreate         - Send Create message to Egress CallAgent
#           McapEvent          - Send Event Message to Ingress or Egress CA
                                 Partner
#           McapDelete         - Send Event Message to Ingress or Egress CA
                                 Partner
#
#           StartTimer         - Request CA EndPointManager to Start timer
#           SequenceError      - Handle event-out-of-sequence condition
#
#   Qualifiers:
#
#           These qualifiers are valid for all Actions.  They indicate if the
#           transition should continue to the next Action or stop if an error
#           is encountered.
#
#                  FatalOnError       stop on error
#                  ContinueOnError    keep processing
#      These qualifiers are used by McapEvent Action to identify the event type.
#
#                  Created            EventType=Create
#                  Answered           EventType=Answer
#                  Released           EventType=Release
#                  Suspended          EventType=Suspend
#                  Resumed            EventType=Resume
#
#      These qualifiers are used by McapCreate Action to identify the Create type.
#
#                  Announce           CreateType=Announce
```

```
#
#          These qualifiers are valid for those Events which can have two flavors.

The other events imply thier Ingress/Egress identity.
#
#                    Ingress        Calling Side
#                    Egress         Called Side
#
#      These qualifiers identify the TimerType being Started by StartTimer Action.
#          #millsec indicates the length of timer in milliseconds.
#
#                    Dial    <#millsec>  Wait for DialComplete message
#                    Create  <#millsec>  Wait for Created message from Egress
#                    Release <#millsec>  Wait to Delete a Connection
#
#      This qualifier is used to identify the Busy condition on the mcapDelete event.
#
#                    Busy    - Line unavailable, in use, busy
#
#          These qualifiers identify the action to take when handing an event
#          sequence error condition.
#
#                    Ignore - Ignore out-of-sequence
#                    Report  - only report out-of-sequence
#                    Reset   - Reset line as Idle and release local resources
#
#          These qualifiers specify the Voip Gateway attributes being passed
#          on CreateConnection, ModifyConnection, DeleteConnection, and
NotifyRequest
#
#                    ReceiveMode          - Put connection in Receive-Only Mode
#                    SendReceiveMode      - Put connection in full-Duplex Mode
#                    SendMode             - Put connection in Send-Only Mode
#                    InactiveMode- Set connection as inactive
#                    LoopBackMode         - ???
#
#                    OpCompleteNotify     - notify CA of Operation Complete
#                    OffHookNotify        - notify CA of Off Hook
#                    OnHookNotify         - notify CA of On Hook
#                    FlashHookNotify      - notify CA of Flash Hook
#                    WinkNotify           - notify CA of Wink
#                    DTMFNotify           - notify CA of DTMF
#                    ContinuityToneNotify
#                    ContinuityDetectedNotify
#                    ModemToneNotify      - notify CA of Modem Tone
#                    FaxToneNotify        - notify CA of Fax Tone
#                    DigitsNotify         - notify CA of Digits collected
#
#                    RingingSignal        - play ringing tone
#                    RingBackSignal       - play ring-back tone
#                    DialToneSignal       - play dial tone
#                    BusyToneSignal       - play Busy tone
#                    CongestionToneSignal - play congestion tone
#                    AnnouncementSignal   - play Announcement
#                    ContinuitySignal     - play continuity tone
#                    CallWaitingSignal    - play call waiting tone
#                    OffHookWarningSignal - play off hook warning tone
#
#=================================================================
==
#------------------------------------------------
# OffHook for CALLING Set-Top-Box
#------------------------------------------------

Event OffHook Ingress

 Transition Idle          Dialing
  Action    NotifyRequest   DigitsNotify DialToneSignal OnHookNotify
  Action    StartTimer      Dial 45

 Transition Active        Active
#------------------------------------------------
# OffHook for CALLED Set-Top-Box
#------------------------------------------------

Event OffHook Egress

 Transition Ringing       Active
  Action    McapEvent       Answered
  Action    NotifyRequest   OnHookNotify

 Transition Active        Active

 Transition Suspended     Active
  Action    McapEvent       Resumed


#
#-----------------------------------------
# DialCompleted - Ingress Only
#-----------------------------------------

Event DialComplete

  Transition Dialing      Initiated
  Action    CreateConnection OnHookNotify  ReceiveMode
  Action    McapCreate
  Action    StartTimer      Create 45

 Transition Idle          Initiated
```

```
    Action   CreateConnection  OnHookNotify   ReceiveMode
    Action   McapCreate
    Action   StartTimer      Create 45

#------------------------------
# Create - Egress Only
#------------------------------

Event Create

  Transition  Idle          Ringing
    Action   CreateConnection  OffHookNotify   RingingSignal  SendReceiveMode
    Action   McapEvent       Created

#-----------------------------
# Busy - Ingress only
#-----------------------------

Event Busy    Ingress

  Transition   Active        Idle
    Action    DeleteConnection  OnHookNotify   OffHookNotify    BusyToneSignal

# Transition   Initiated      Idle
  Transition   Initiated      *
    Action    DeleteConnection  OnHookNotify   OffHookNotify    BusyToneSignal

  Transition   Delivered      Idle
    Action    DeleteConnection  OnHookNotify   OffHookNotify    BusyToneSignal




#----------------------------
#    —
# Busy - Egress Only
#----------------------------

Event Busy    Egress

  Transition *             *
    Action   McapDelete      Busy

#
#-----------------------------------------
# Announcement - Ingress only
#-----------------------------------------

Event Announcement  Ingress
```

Transition Action	* McapCreate	* Announce

```
#
#------------------------------------------
# Announcement - Egress only
#------------------------------------------

Event Announcement  Egress

 Transition  *              *
  Action   CreateConnection  OpCompleteNotify AnnouncementSignal  SendMode

#
#------------------------------------
# InvalidEndpoint - Ingress
#------------------------------------

Event InvalidEndpoint  Ingress

 Transition   Active        Idle
  Action    DeleteConnection     OnHookNotify  OffHookNotify  BusyToneSignal

 Transition   Initiated     Idle
  Action    DeleteConnection     OnHookNotify  OffHookNotify  BusyToneSignal

 Transition   Delivered      Idle
  Action    DeleteConnection     OnHookNotify  OffHookNotify  BusyToneSignal

#
#------------------------------------
# InvalidEndpoint - Egress
#------------------------------------

Event InvalidEndpoint  Egress

 Transition  *              *
  Action   McapDelete      InvalidEndpoint
```

```
#
#--------------------------------
# Created - Ingress Only
#--------------------------------

Event Created

 Transition Dialing       Delivered

 Transition Initiated     Delivered
#  Action   ModifyConnection OnHookNotify  RingBackSignal ReceiveMode
   Action   ModifyConnection OnHookNotify  RingBackSignal SendReceiveMode

#
#-----------------------------------
# Answered - Ingress Only
#-----------------------------------

Event Answered

 Transition Idle          Active

 Transition Initiated     Active
#  Action   ModifyConnection OnHookNotify  SendReceiveMode
   Action   NotifyRequest    OnHookNotify
   Action   AccountingStart

 Transition Delivered     Active
#  Action   ModifyConnection OnHookNotify  SendReceiveMode
   Action   NotifyRequest    OnHookNotify
   Action   AccountingStart

#
#----------------------------------------------
# OnHook for CALLING Set-Top-Box
#----------------------------------------------

Event OnHook  Ingress

 Transition  Releasing     Idle
  Action     NotifyRequest    OnHookNotify  OffHookNotify

 Transition  Active        Releasing
  Action     McapDelete
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     StartTimer       Release 45

 Transition  Dialing       Idle
  Action     NotifyRequest    OnHookNotify  OffHookNotify
 Transition  Initiated     Releasing
  Action     McapDelete
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     StartTimer      Release 45

 Transition  Delivered     Releasing
  Action     McapDelete
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     StartTimer      Release 45

 Transition  *             Idle
  Action     NotifyRequest   OnHookNotify  OffHookNotify

#
#----------------------------------------------
# OnHook for CALLED Set-Top-Box
#----------------------------------------------

Event OnHook  Egress

 Transition  Releasing     Idle
  Action     NotifyRequest   OnHookNotify  OffHookNotify

 Transition  Active        Suspended
  Action     NotifyRequest   OnHookNotify  OffHookNotify
  Action     StartTimer      Release 15
  Action     McapEvent       Suspended

 Transition  *             Idle
  Action     NotifyRequest   OnHookNotify  OffHookNotify
```

```
#
#------------------------------------------------
# Delete for CALLING Set-Top-Box
#------------------------------------------------

Event Delete   Ingress

 Transition  Active         Idle
  Action     McapEvent       Released
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     AccountingStop

 Transition  Idle           Idle
  Action     McapEvent       Released
  Action     NotifyRequest   OnHookNotify  OffHookNotify

 Transition  Dialing        Dialing
  Action     McapEvent       Released

 Transition  Releasing      Idle
  Action     NotifyRequest   OnHookNotify  OffHookNotify
  Action     AccountingStop

 Transition  Initiated      Idle
  Action     McapEvent       Released
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     AccountingStop

 Transition  Delivered      Idle
  Action     McapEvent       Released
  Action     DeleteConnection OnHookNotify  OffHookNotify
  Action     AccountingStop

#
#------------------------------------------------
# Delete for CALLED Set-Top-Box
#------------------------------------------------

Event Delete   Egress

 Transition  Active         Idle
  Action     McapEvent       Released
  Action     DeleteConnection OnHookNotify  OffHookNotify

 Transition  Suspended      Idle
  Action     McapEvent       Released
  Action     DeleteConnection OnHookNotify  OffHookNotify

 Transition  Idle           Idle
  Action     McapEvent       Released
  Action    NotifyRequest   OnHookNotify  OffHookNotify

 Transition  Ringing       Idle
  Action    McapEvent      Released
  Action    DeleteConnection OnHookNotify  OffHookNotify

 Transition  Releasing      Idle
  Action    NotifyRequest   OnHookNotify  OffHookNotify

#
#--------------------------------------------------
# Released for CALLING Set-Top-Box
#--------------------------------------------------

Event Released  Ingress

 Transition  Releasing      Idle
  Action    AccountingStop
  Action    NotifyRequest   OnHookNotify  OffHookNotify

 Transition  *            *

  Action    AccountingStop

#
#--------------------------------------------------
# Released for CALLED Set-Top-Box
#--------------------------------------------------

Event Released  Egress

 Transition  Releasing      Idle
  Action    NotifyRequest   OnHookNotify  OffHookNotify

 Transition  Suspended      Idle
  Action    NotifyRequest   OnHookNotify  OffHookNotify

 Transition  *            *



#
#-----------------------------------------------------------
# Expired Timer waiting for digits to be dialed
#-----------------------------------------------------------

Event TimerExpired Dial

 Transition  Dialing       Idle
  Action    NotifyRequest   OnHookNotify  OffHookNotify   DialToneSignal
```

```
#
#-----------------------------------------------------
#  Expired Timer waiting for Create ACK
#-----------------------------------------------------

Event TimerExpired Create

 Transition  Initiated       Idle
   Action    McapDelete
   Action    DeleteConnection OnHookNotify  OffHookNotify

#
#------------------------------------------------------------------
#  Expired Timer waiting for hang-up delay interval
#------------------------------------------------------------------

Event TimerExpired Release

 Transition  Suspended       Releasing
   Action    McapDelete
   Action    DeleteConnection OnHookNotify  OffHookNotify

 Transition  Releasing      Idle
   Action    NotifyRequest  OnHookNotify  OffHookNotify
```

WHAT IS CLAIMED IS:

1. A communications system, comprising:

a packet-based network;

a first subscriber unit;

a first media control device connecting the first subscriber unit to the packet-based network;

a second subscriber unit;

a second media control device connecting the second subscriber unit to the packet-based network; and

a call agent comprising;

means for managing communications between the first and second subscriber units over the network, and

means for sending and/or receiving SS7 signaling information.

2. The system of claim 1, wherein the first subscriber unit is coupled to the packet-based network through a public switched telephone network.

3. The system of claim 1, wherein the packet-based network is an Internet protocol network.

4. The system of claim 1, wherein the packet-based network is an asynchronous transfer mode network.

5. The system of claim 1, wherein the call agent further comprises: means for transmitting information to a media control device.

6. The system of claim 5, wherein the information transmitted to the media control device is in the simple gateway control protocol.

7. The system of claim 5, wherein the media control device is an Internet protocol gateway.

8. A communications system, comprising:

a first subscriber unit coupled to a network through a first media control device;

a second subscriber unit coupled to the network through a second media control device; and

a call agent, comprising:

a first call agent cluster coupled to the first subscriber unit through a media control device, comprising:

means for translating information received from the first media control device in a first protocol into a common protocol,

means for communicating with a second call agent cluster using the common protocol,

means for translating the information in the common protocol into the first protocol, and

means for controlling the first media control device for managing a media session between the first subscriber unit and the second subscriber unit over the network.

9. The system of claim 8, wherein the first media control device is a residential gateway.

10. The system of claim 8, further comprising:

a third media control device, wherein the third media control device is a trunking gateway; and

wherein the first call agent comprises means for controlling the third media control device; and

wherein the first media control device is an SS7 gateway;

11. The system of claim 8, wherein the first media control device is an H.323 gateway.

12. The system of claim 8, wherein the call agent further comprises:

a service broker, comprising:

means for receiving information from the first call agent cluster;

means for determining the second call agent cluster from a plurality of call agent clusters;

means for sending information to the second call agent cluster regarding setting up communications with the first call agent cluster; and

wherein the first call agent cluster further comprises means for transmitting information to the service broker.

13. The system of claim 12, wherein the call agent further comprises:

a network resource database, comprising:

means for storing data;

means for receiving requests from the service broker; and

means for transmitting information to the service broker

14. The system of claim 8, wherein the network is an IP network.

15. The system of claim 8, wherein the network is an ATM network.

16. The system of claim 8, wherein the first call agent cluster and second call agent cluster communicate over a CORBA software bus.

17. The system of claim 8, further comprising:

an accounting gateway, comprising:

means for sending information to a billing system.

18. The system of claim 8, further comprising:

an announcement server, comprising:

means for playing pre-recorded messages.

19. The system of claim 8, wherein the common protocol is the multi call agent protocol.

20. The system of claim 8, wherein the first call agent cluster, comprises:

an endpoint manager; and

a state machine;

wherein the endpoint manager comprises:

means for storing information on a media session, and

means for transmitting information to the state machine, and

wherein the state machine comprises:

means for receiving information from the endpoint manager, and

means for using the information to take an action.

21. The system of claim 20, wherein the first call agent cluster further comprises:

a message queue, comprising:

means for receiving information from a subscriber unit,

means for storing the information, and

means for transmitting information to the endpoint manager.

22. The system of claim 8, wherein the first call agent cluster further comprises:

a gateway object, comprising:

means for communicating with the state machine, and

means for managing the first gateway.

23. The system of claim 8, wherein the first call agent cluster further comprises:

a message handler, comprising:

means for determining an endpoint manager from a plurality of endpoint managers,

means for receiving information, and

means for transmitting the information to the determined endpoint manager.

24. The system of claim 8, further comprising:

a switch linking the first subscriber unit to the first media control device.

25. The system of claim 8, wherein the call agent further comprises:

means for embedding in the common protocol the information in the first protocol.

26. A method of managing communications between a first subscriber unit and a second subscriber unit over a network, comprising the steps of:

the call agent sending and/or receiving SS7 signaling information regarding management of communications over a packet-based network;

the call agent managing communications between the first and second subscriber units over the network; and

the first and second subscriber units communicating over the network.

27. The method of 26, wherein the first subscriber unit is connected to the call agent through a public switched telephone network.

28. The method of 26, wherein the packet-based network is an internet protocol network.

29. The method of 26, wherein the packet-based network is an asynchronous transfer mode network.

30. The method of 26, further comprising the step of the call agent transmitting information to a media control device.

31. The method of 30, wherein the information transmitted to the media control device is in the simple gateway control protocol.

32. The method of 30, wherein the media control device is an internet protocol gateway.

33. A method of managing communications between a first subscriber unit and a second subscriber unit, comprising the steps of:

a first media control device coupled to the first subscriber unit transmitting information in a first protocol to a first call agent cluster regarding establishing a media session with the second subscriber unit over a packet-based network;

the first call agent cluster translating the information in the first protocol to a common protocol;

setting up a connection between the first call agent cluster and a second call agent cluster;

the first call agent cluster and the second call agent cluster exchanging information using the common protocol;

the first call agent cluster translating information in the common protocol to the first protocol;

the first call agent cluster transmitting the information in the first protocol to the first media control device coupled to the first subscriber unit;

the second call agent cluster translating information in the common protocol to a second protocol;

the second call agent cluster transmitting the information in the second protocol to a second media control device coupled to the second subscriber unit; and

the first subscriber unit and the second subscriber unit exchanging information over the network.

34. The method of claim 33, wherein the first media control device is a residential gateway.

35. The method of claim 33, wherein the first media control device is an H.323 gateway.

36. The method of claim 33, further comprising the step of

the first call agent cluster transmitting information to a third media control device wherein the third media control device is a trunking gateway; and

wherein the first media control device is an SS7 gateway.

37. The method of claim 33, wherein the step of setting up a connection between the first call agent cluster and the second call agent cluster comprises the sub-steps of:

the first call agent cluster transmitting information to a service broker;

the service broker determining the second call agent cluster from a plurality of call agent clusters;

the service broker transmitting information to the second call agent cluster.

38. The method of claim 37, further comprising the sub-steps of:

the service broker transmitting a request to a network resource database; and

the network resource database transmitting information to the service broker.

39. The method of claim 33, wherein the network is an IP network.

40. The method of claim 33, wherein the network is an ATM network.

41. The method of claim 33, wherein the call agent clusters communicate over a CORBA bus.

42. The method of claim 33, wherein the call agent clusters communicate using the multi call agent protocol.

43. The method of claim 33, wherein the information in the first protocol is embedded in the information in the common protocol.

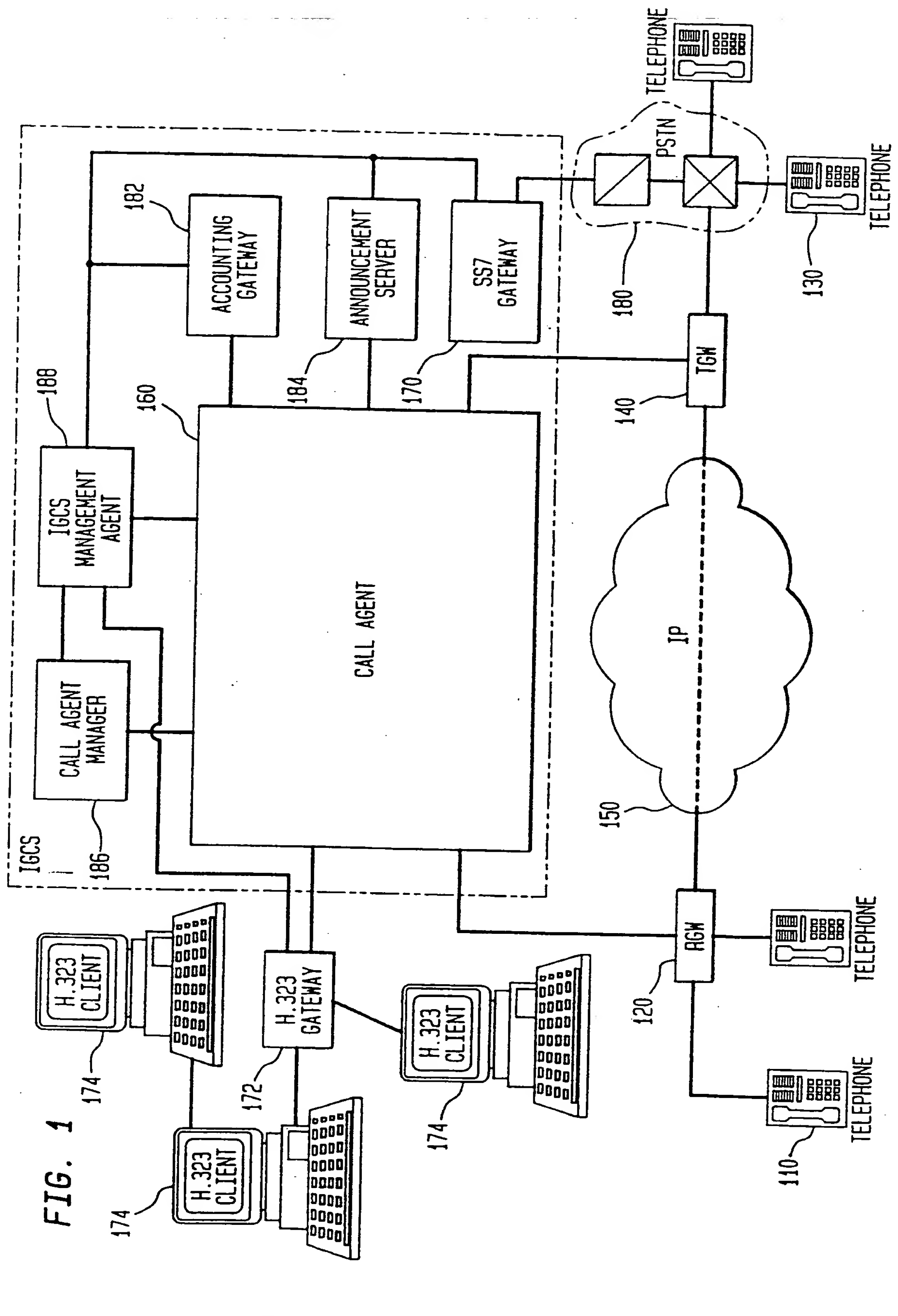
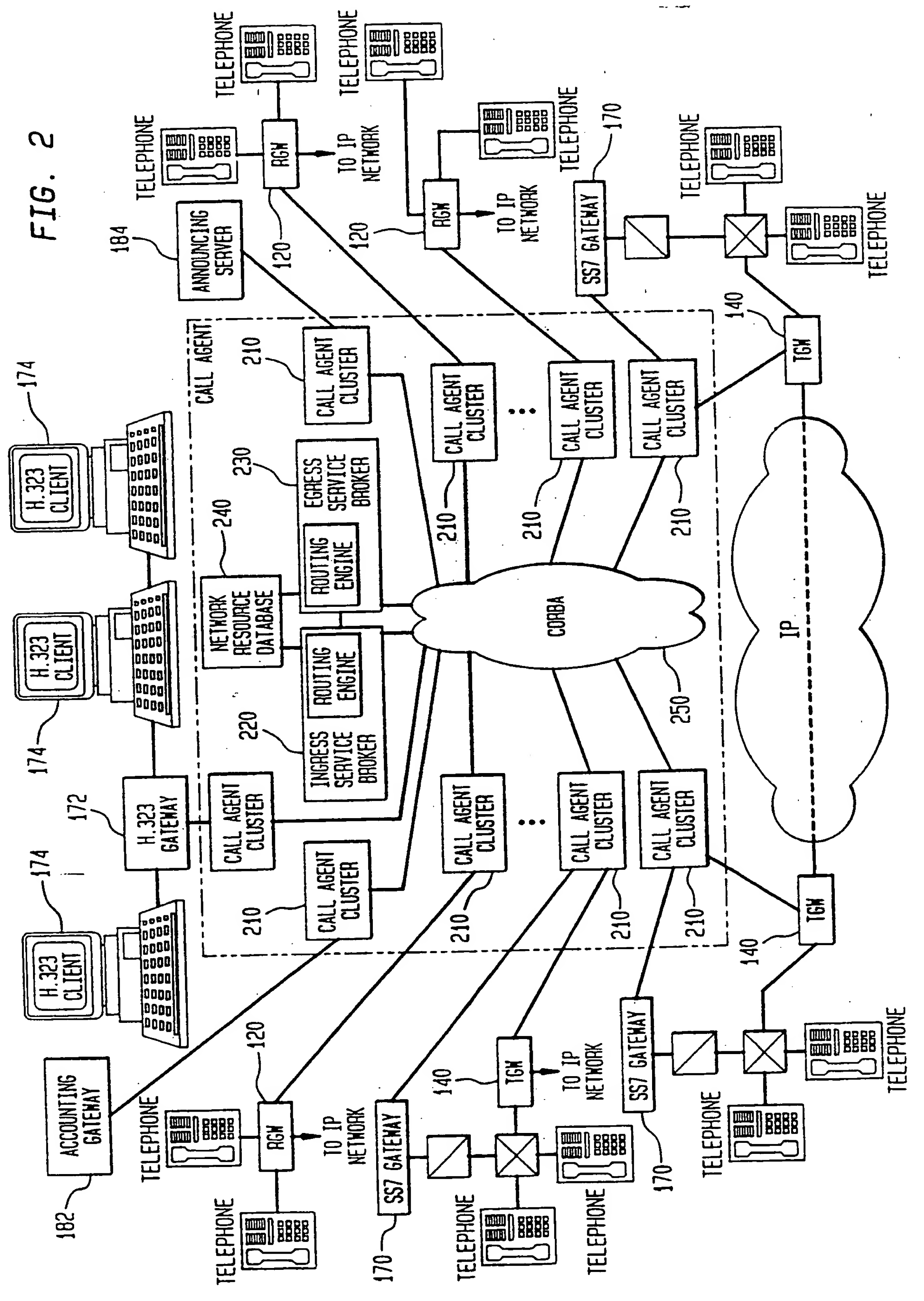

TELEPHONE
TELEPHONE
130
PSTN
180
TGW
140
IP
150
RGW
120
TELEPHONE
TELEPHONE
110
IGCS
ACCOUNTING GATEWAY
182
ANNOUNCEMENT SERVER
184
SS7 GATEWAY
170
160
CALL AGENT
IGCS MANAGEMENT AGENT
188
CALL AGENT MANAGER
186
H.323 GATEWAY
172
H.323 CLIENT
174
H.323 CLIENT
174
H.323 CLIENT
174

FIG. 1

FIG. 2


TELEPHONE
RGW
TO IP NETWORK
120
ANNOUNCING SERVER
184
SS7 GATEWAY
170
TGW
140
CALL AGENT
CALL AGENT CLUSTER
210
EGRESS SERVICE BROKER
230
ROUTING ENGINE
NETWORK RESOURCE DATABASE
240
INGRESS SERVICE BROKER
220
CORBA
250
IP
H.323 CLIENT
174
H.323 GATEWAY
172
ACCOUNTING GATEWAY
182

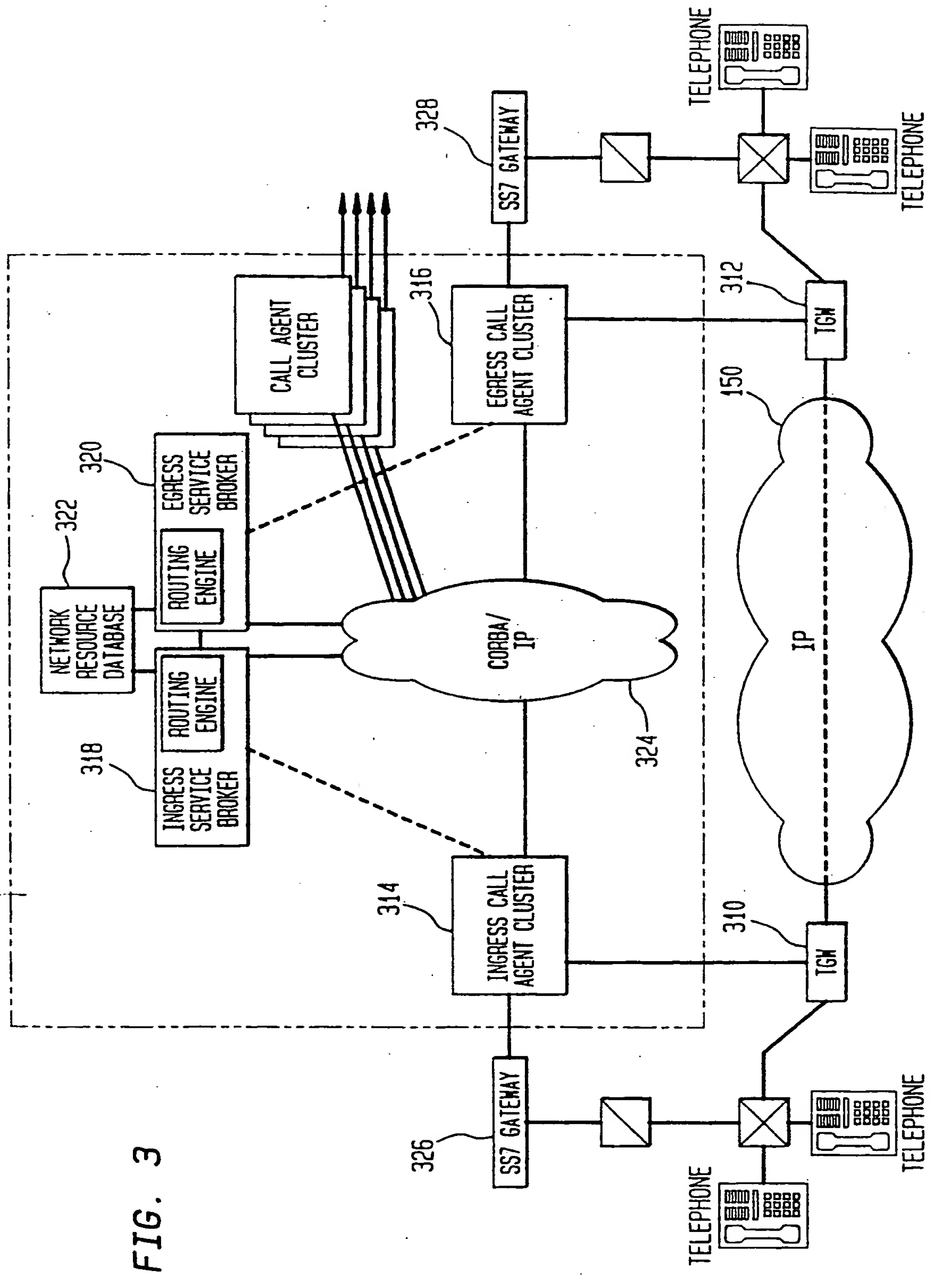

TELEPHONE
TELEPHONE
TELEPHONE
TELEPHONE
SS7 GATEWAY
SS7 GATEWAY
328
326
TGW
TGW
312
310
150
IP
316
EGRESS CALL AGENT CLUSTER
314
INGRESS CALL AGENT CLUSTER
CALL AGENT CLUSTER
320
EGRESS SERVICE BROKER
ROUTING ENGINE
318
INGRESS SERVICE BROKER
ROUTING ENGINE
322
NETWORK RESOURCE DATABASE
CORBA/IP
324

FIG. 3

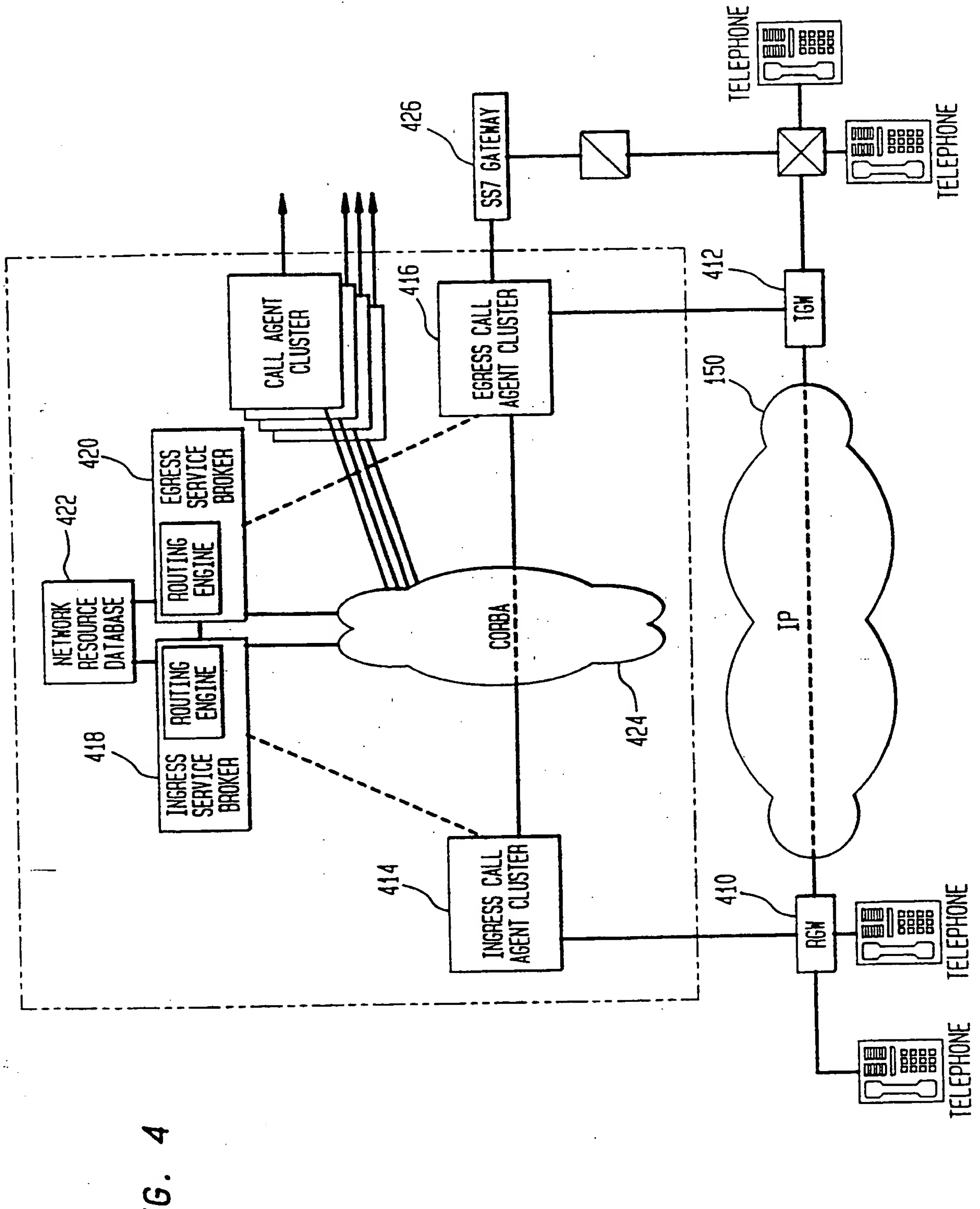

CALL AGENT CLUSTER
426
SS7 GATEWAY
TELEPHONE
TELEPHONE
420
EGRESS SERVICE BROKER
ROUTING ENGINE
422
NETWORK RESOURCE DATABASE
416
EGRESS CALL AGENT CLUSTER
412
TGW
150
IP
CORBA
424
ROUTING ENGINE
418
INGRESS SERVICE BROKER
414
INGRESS CALL AGENT CLUSTER
410
RGW
TELEPHONE
TELEPHONE

FIG. 4

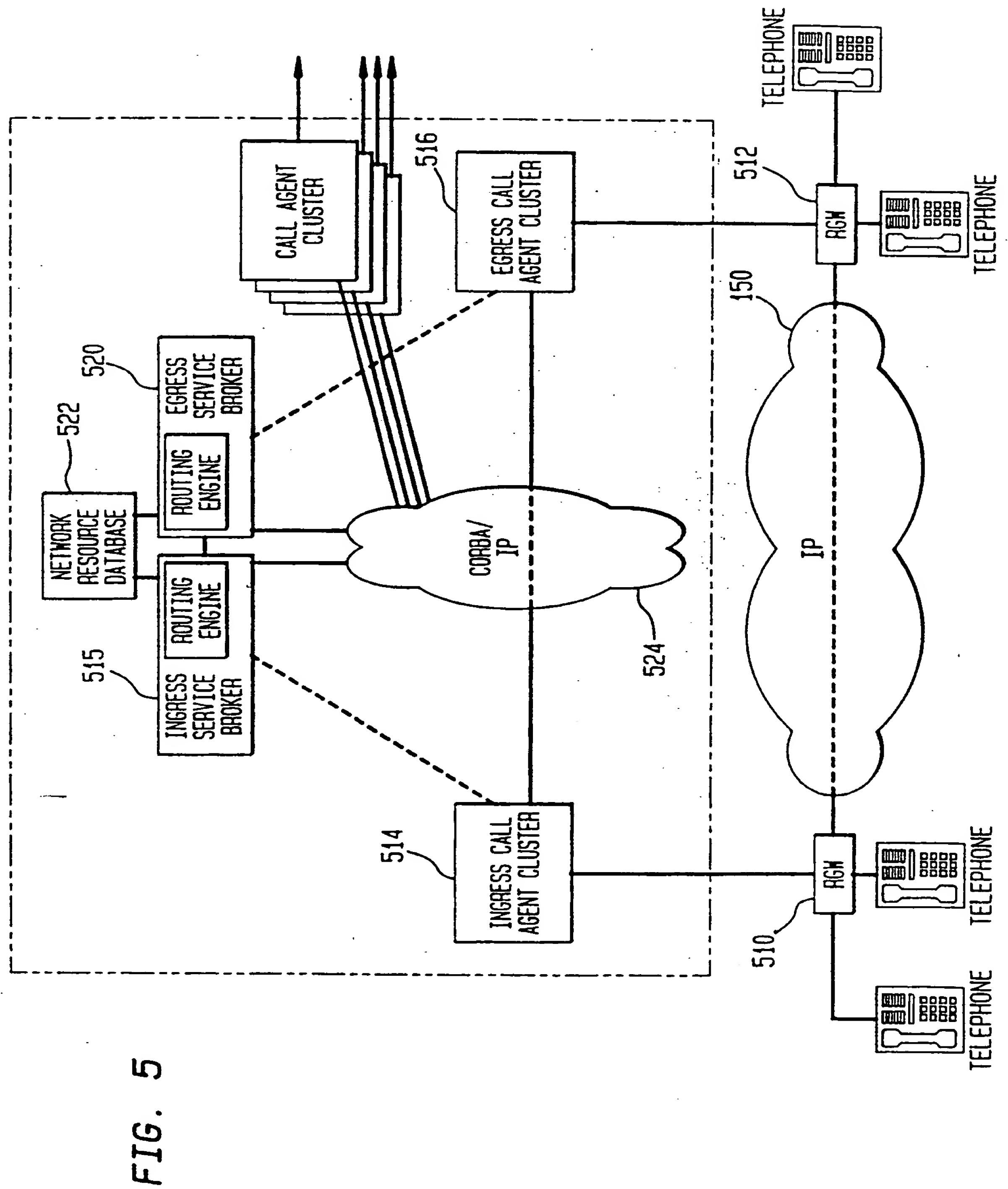
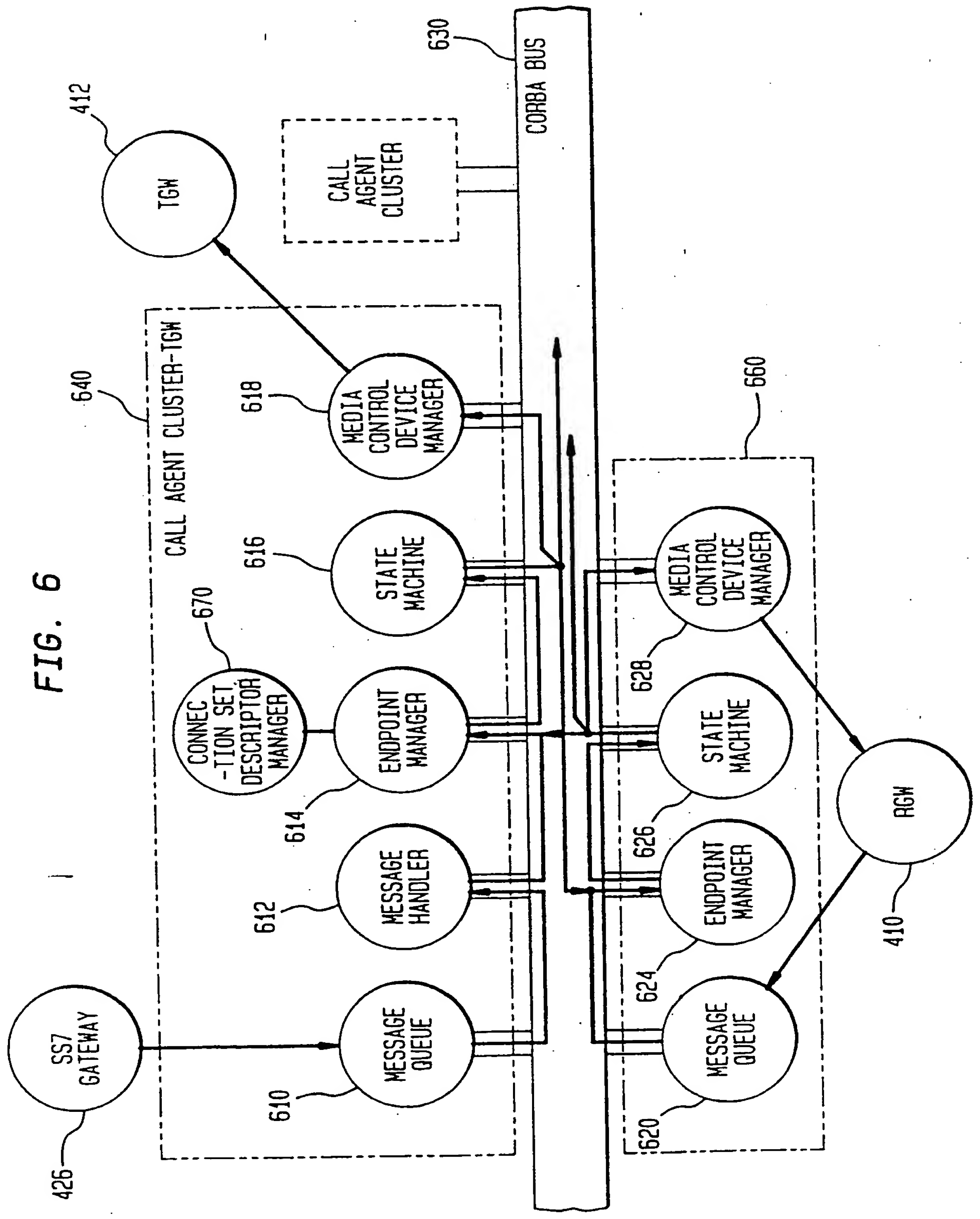

CALL AGENT CLUSTER
516
EGRESS CALL AGENT CLUSTER
512
RGW
TELEPHONE
TELEPHONE
520
EGRESS SERVICE BROKER
ROUTING ENGINE
522
NETWORK RESOURCE DATABASE
ROUTING ENGINE
515
INGRESS SERVICE BROKER
CORBA/ IP
524
150
IP
514
INGRESS CALL AGENT CLUSTER
RGW
510
TELEPHONE
TELEPHONE

FIG. 5

FIG. 6


630
CORBA BUS
412
TGW
CALL AGENT CLUSTER
640
CALL AGENT CLUSTER-TGW
618
MEDIA CONTROL DEVICE MANAGER
616
STATE MACHINE
670
CONNEC -TION SET DESCRIPTOR MANAGER
ENDPOINT MANAGER
614
612
MESSAGE HANDLER
610
MESSAGE QUEUE
SS7 GATEWAY
426
660
MEDIA CONTROL DEVICE MANAGER
628
STATE MACHINE
626
ENDPOINT MANAGER
624
MESSAGE QUEUE
620
RGW
410

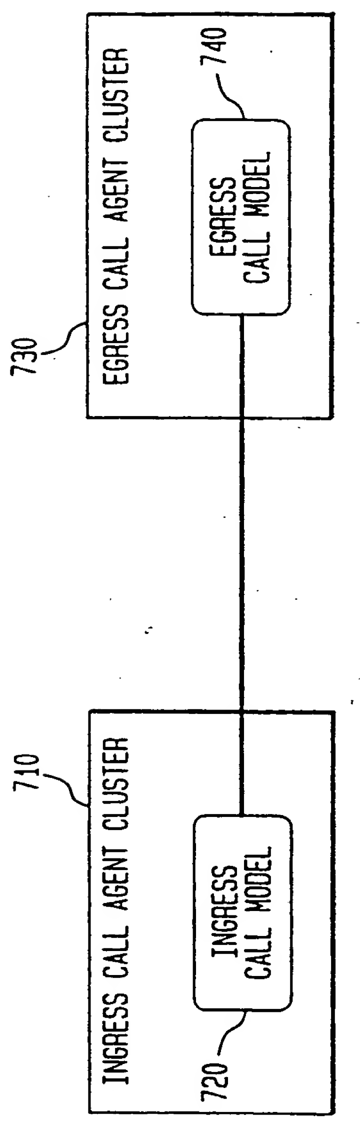

INGRESS CALL AGENT CLUSTER
710
INGRESS CALL MODEL
720
EGRESS CALL AGENT CLUSTER
730
EGRESS CALL MODEL
740

FIG. 7

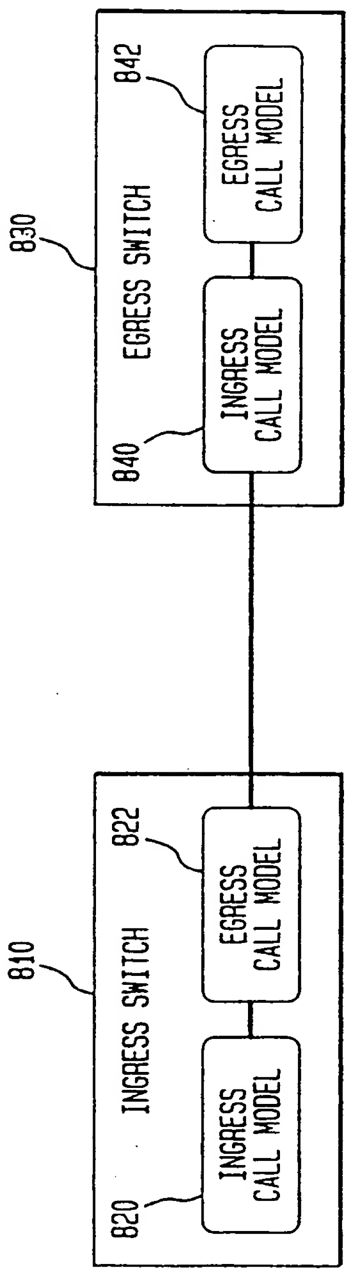

INGRESS SWITCH
810
INGRESS CALL MODEL
820
EGRESS CALL MODEL
822
EGRESS SWITCH
830
INGRESS CALL MODEL
840
EGRESS CALL MODEL
842

FIG. 8

FIG. 9

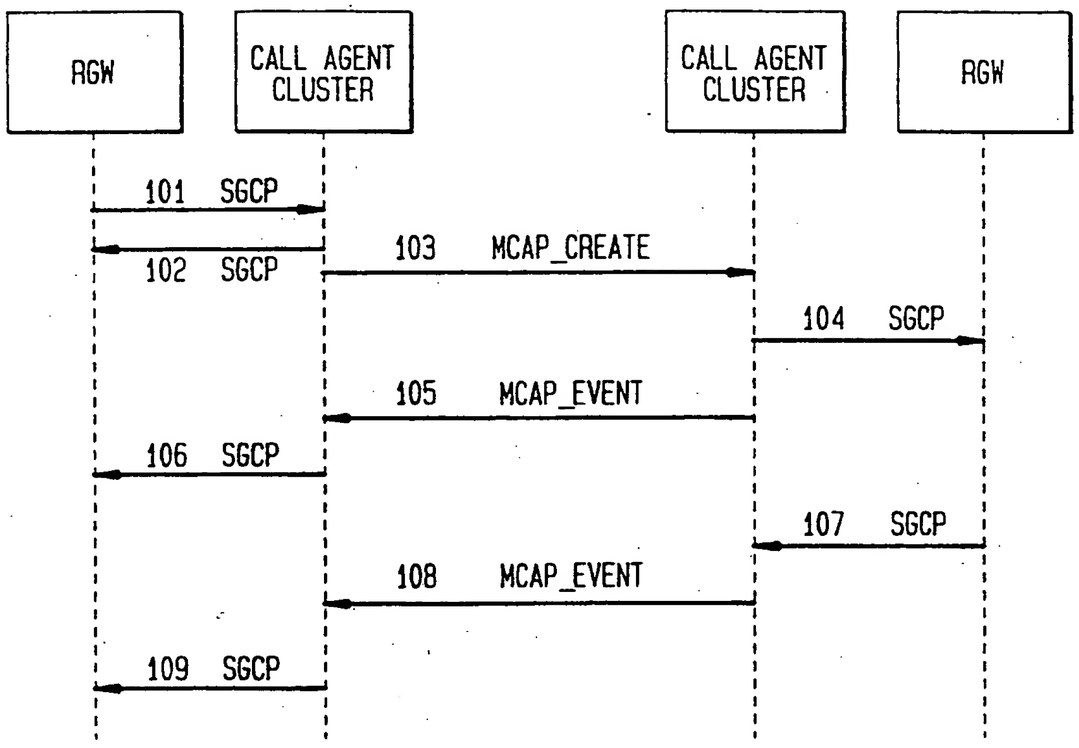

RGW
CALL AGENT CLUSTER
CALL AGENT CLUSTER
RGW
101 SGCP
102 SGCP
103 MCAP_CREATE
104 SGCP
105 MCAP_EVENT
106 SGCP
107 SGCP
108 MCAP_EVENT
109 SGCP

FIG. 10

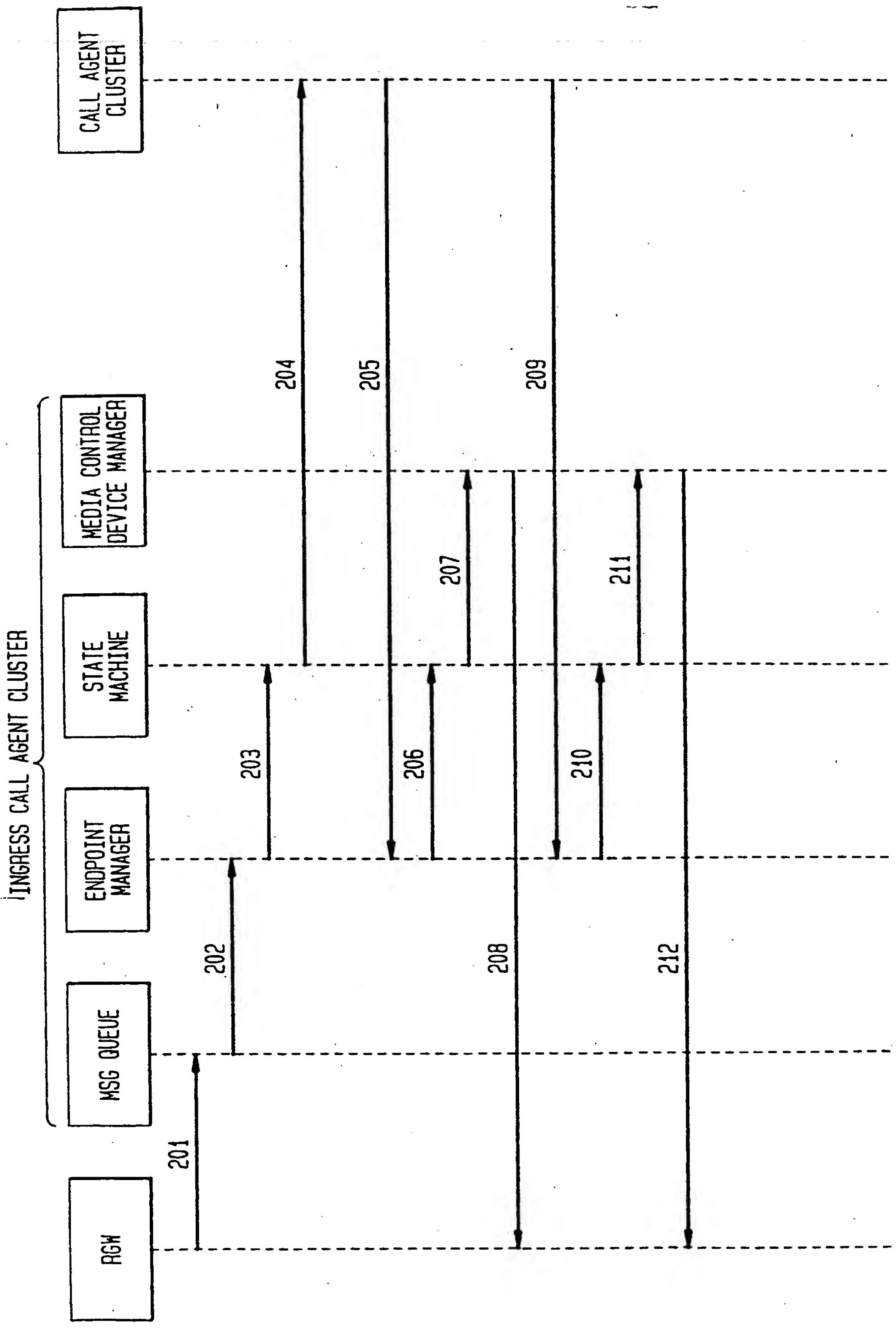

CALL AGENT CLUSTER
INGRESS CALL AGENT CLUSTER
MEDIA CONTROL DEVICE MANAGER
STATE MACHINE
ENDPOINT MANAGER
MSG QUEUE
RGW
201
202
203
204
205
206
207
208
209
210
211
212

FIG. 11

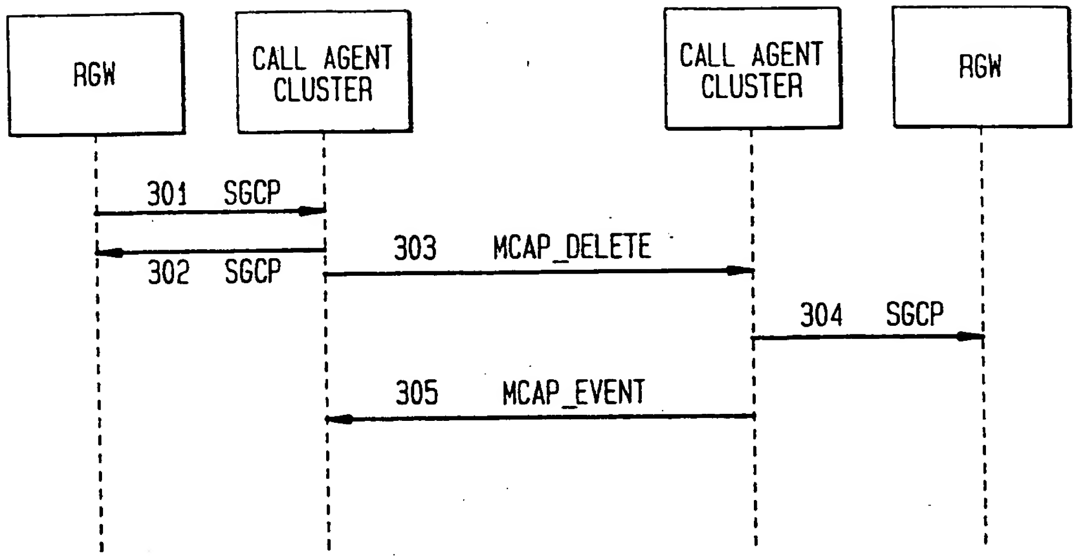

RGW
CALL AGENT CLUSTER
CALL AGENT CLUSTER
RGW
301 SGCP
302 SGCP
303 MCAP_DELETE
304 SGCP
305 MCAP_EVENT

FIG. 12

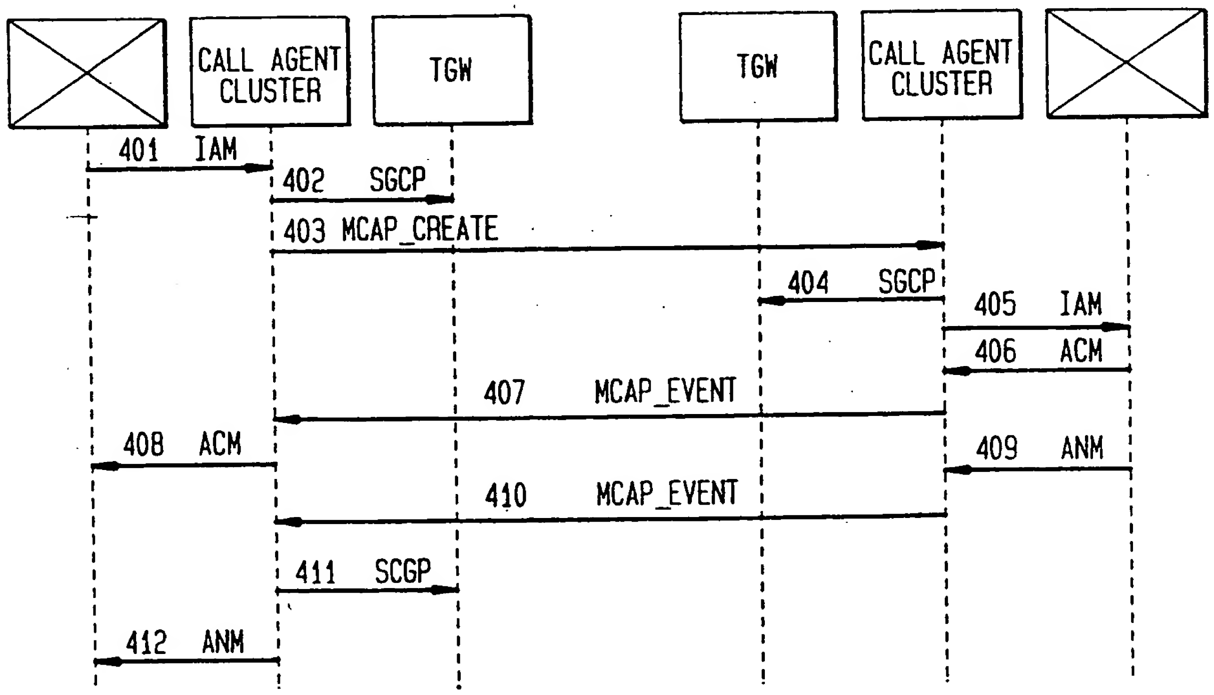
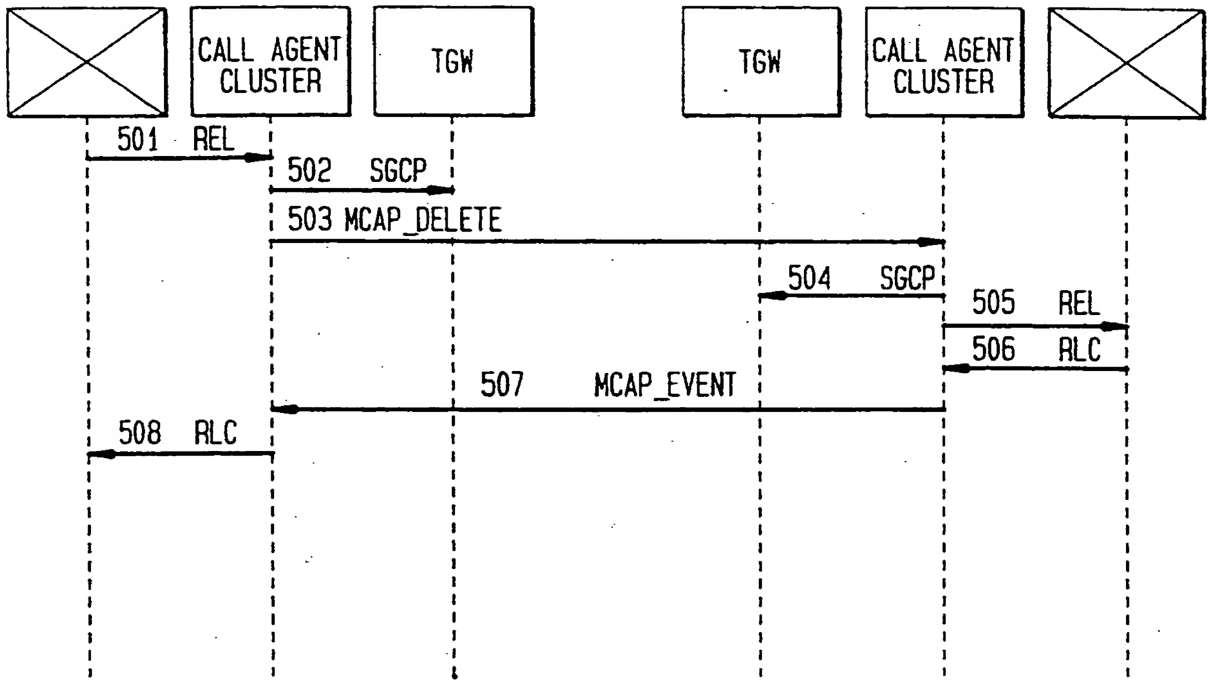

CALL AGENT CLUSTER
TGW
TGW
CALL AGENT CLUSTER
401 IAM
402 SGCP
403 MCAP_CREATE
404 SGCP
405 IAM
406 ACM
407 MCAP_EVENT
408 ACM
409 ANM
410 MCAP_EVENT
411 SCGP
412 ANM

FIG. 13



CALL AGENT CLUSTER
TGW
TGW
CALL AGENT CLUSTER
501 REL
502 SGCP
503 MCAP_DELETE
504 SGCP
505 REL
506 RLC
507 MCAP_EVENT
508 RLC

FIG. 14

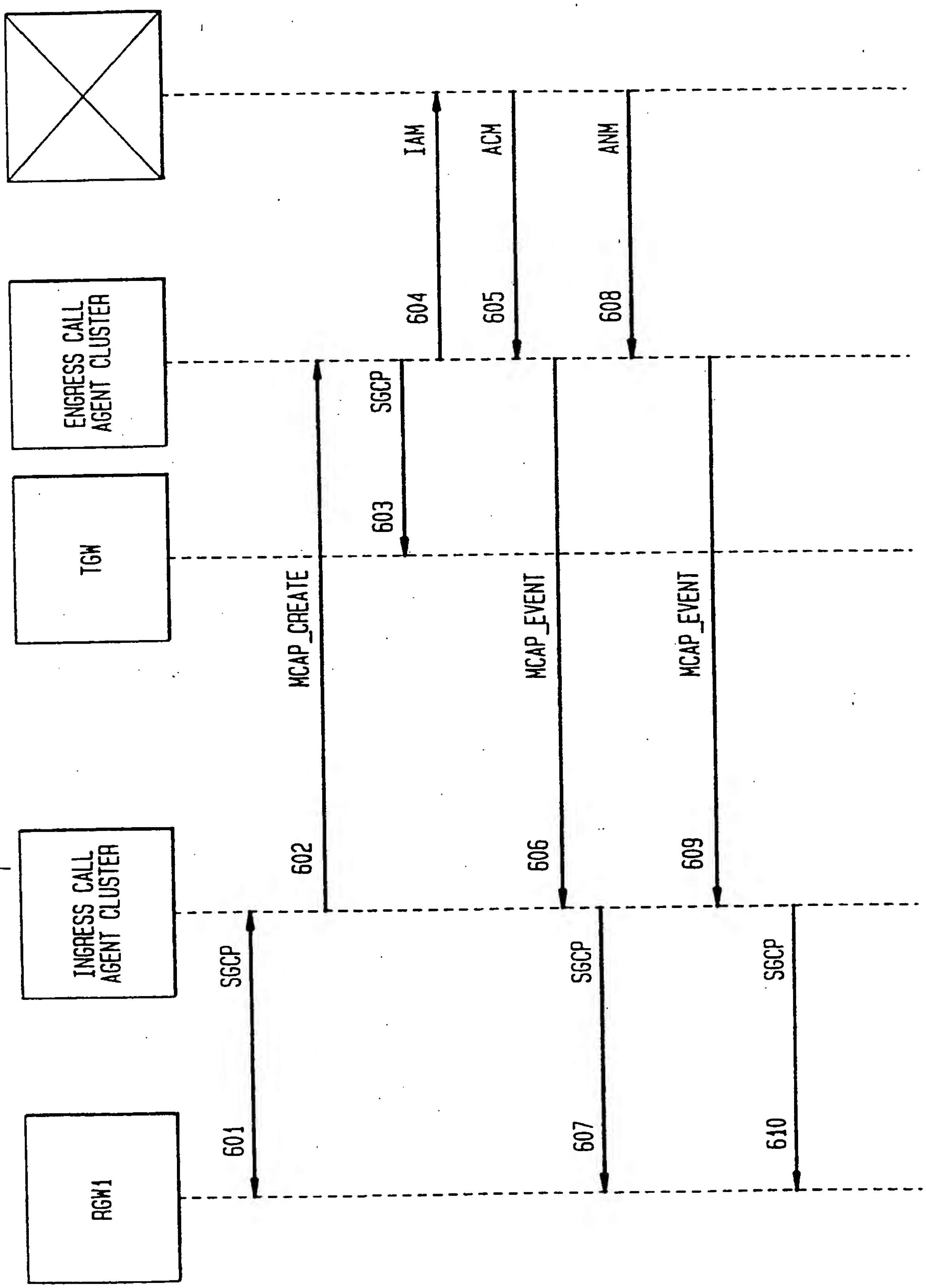

RGW1
INGRESS CALL AGENT CLUSTER
TGW
ENGRESS CALL AGENT CLUSTER
SGCP
601
MCAP_CREATE
602
SGCP
603
IAM
604
ACM
605
MCAP_EVENT
606
SGCP
607
ANM
608
MCAP_EVENT
609
SGCP
610

FIG. 15

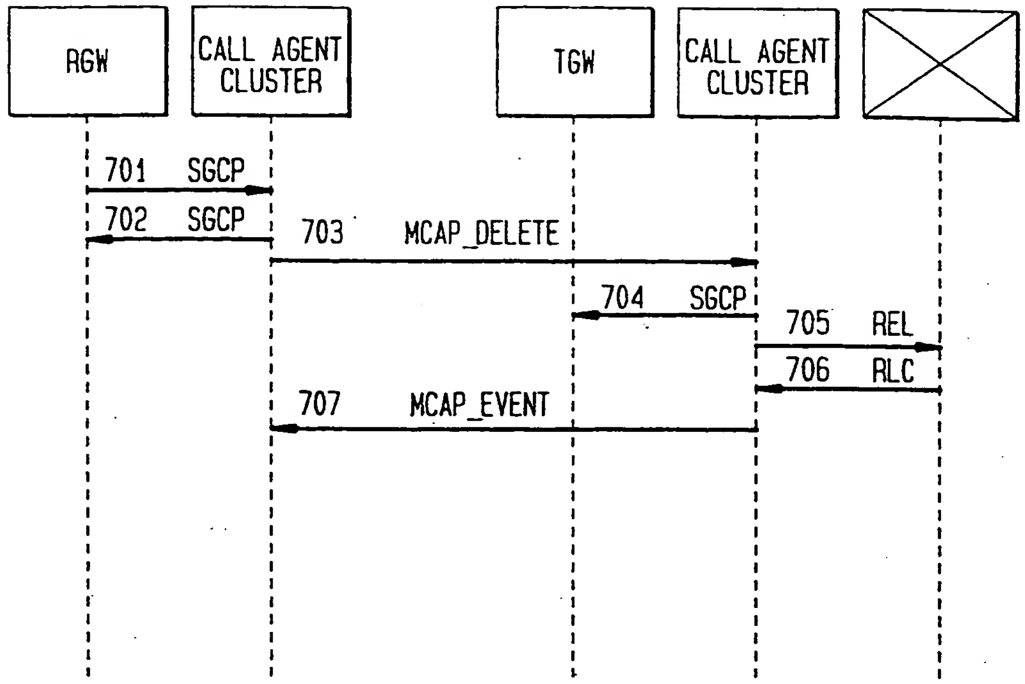

RGW
CALL AGENT CLUSTER
TGW
CALL AGENT CLUSTER
701 SGCP
702 SGCP
703 MCAP_DELETE
704 SGCP
705 REL
706 RLC
707 MCAP_EVENT

FIG. 16

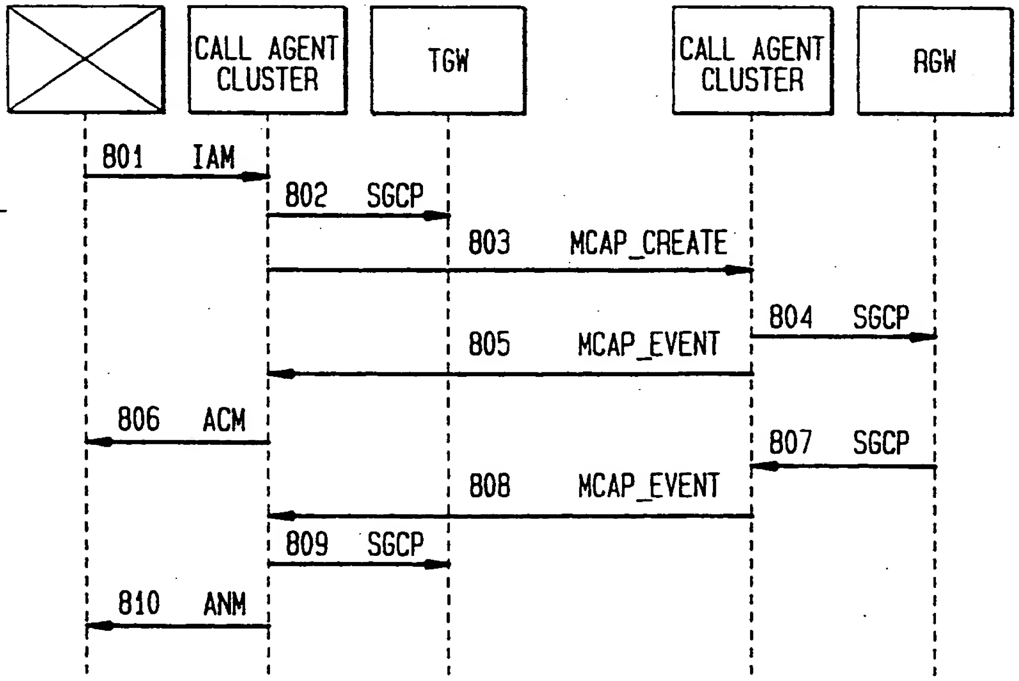
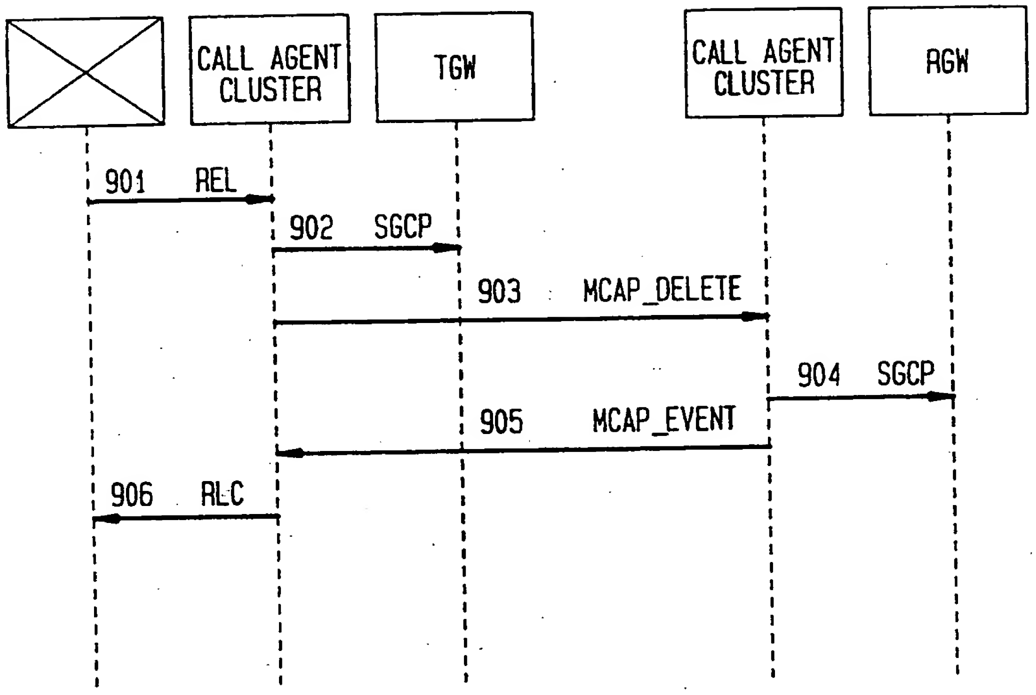

CALL AGENT CLUSTER
TGW
CALL AGENT CLUSTER
RGW
801 IAM
802 SGCP
803 MCAP_CREATE
804 SGCP
805 MCAP_EVENT
806 ACM
807 SGCP
808 MCAP_EVENT
809 SGCP
810 ANM

FIG. 17


CALL AGENT CLUSTER
TGW
CALL AGENT CLUSTER
RGW
901 REL
902 SGCP
903 MCAP_DELETE
904 SGCP
905 MCAP_EVENT
906 RLC

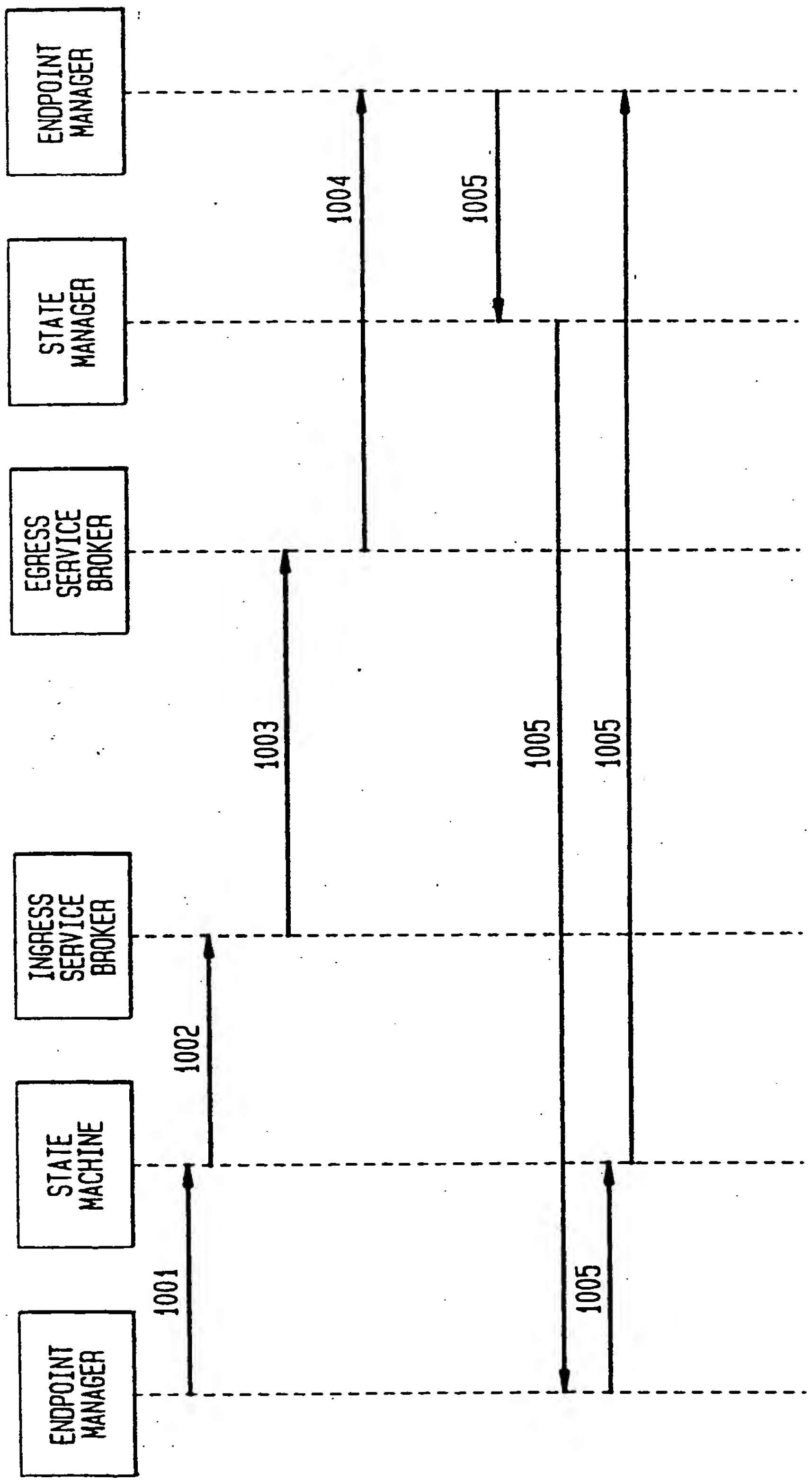

ENDPOINT MANAGER
STATE MACHINE
INGRESS SERVICE BROKER
EGRESS SERVICE BROKER
STATE MANAGER
ENDPOINT MANAGER
1001
1002
1003
1004
1005
1005
1005
1005
1005

FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/25760

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 5,581,596A (HOGAN) 03 December 1996, col. 3-4, 7-8, and fig. 1	1-3, 8, 20, 21, 26, 27, 33
Y, P	US 5,706,286A (REIMAN et al.) 06 January 1998, col. 2-3, 4-6, 39-40, and fig. 1	1-3, 5-11, 14, 15, 17-28, 30-39, 40, 42, 43
Y, P	US 5,764,750A (CHAU et al.) 09 June 1998, col. 1-4, 17-18, 26-27, and abstract	1-3, 8, 11-13, 16, 26, 27, 33, 35, 41

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US98/25760

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 13/00
US CL :709/223, 227
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. :

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS
search terms: manage communication, call agent, packet network, media control, SS7 gateway, VOIP, SGCP, ATM, convert protocol

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 4,656,624A (COLLINS et al.) 07 April 1987, col. 3-5, 14-15, and abstract	1, 2, 8, 26, 27, 33
Y	US 4,949,373A (BAKER, JR. et al.) 14 August 1990, col. 1-2, 8, 18 and abstract	1, 2, 8, 16-18, 20, 21, 26, 27, 33, 41
Y	US 5,329,619A (PAGE et al.) 12 July 1994, see entire document especially col. 3, 53-54 and abstract	1-8, 11-14, 19-33, 35-39, 42-43
Y	US 5,550,906A (CHAU et al.) 27 August 1996, col. 2-3, 5-6, 18-19, and abstract	1-5, 8-10, 14, 15, 19-21, 24-30, 33, 34, 37-40

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date		
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed	"&"	document member of the same patent family

Date of the actual completion of the international search 23 MARCH 1999	Date of mailing of the international search report 09 APR 1999
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer JASON D. CARDONE James R. Matthews Telephone No. (703) 305-3800